Towards Open Scan for the Open-source Hardware

Leonid Azriel
Department of Computer Science
Technion - Israel Institute of Technology
Haifa, Israel
leonida@technion.ac.il

Avi Mendelson
Department of Computer Science
Technion - Israel Institute of Technology
Haifa, Israel
avi.mendelson@technion.ac.il

***Abstract*—The open-source hardware IP model has recently started gaining popularity in the developer community. This model offers the integrated circuit (IC) developers wider standardization, faster time-to-market and richer platform for research. In addition, open-source hardware conforms to the Kerckhoff's principle of a publicly-known algorithm and thus helps to enhance security. However, when security comes into consideration, source transparency is only one part of the solution. A complex global IC supply chain stands between the source and the final product. Hence, even if the source is known, the finished product is not guaranteed to match it. In this article, we propose the Open Scan model, in which, in addition to the source code, the IC vendor contributes a library-independent information on scan insertion. With scan information available, the user or a certification lab can perform partial reverse engineering of the IC to verify conformance to the advertised source. Compliance lists of open-source programs, such as of the OpenTitan cryptographic IC, can be amended to include this requirement. The Open Scan model addresses accidental and dishonest deviations from the golden model and partially addresses malicious modifications, known as hardware Trojans. We verify the efficiency of the proposed method in simulation with the Trust-Hub Trojan benchmarks and with several open-source benchmarks, in which we randomly insert modifications.**

I. Introduction

The software community, both academia and commercial enterprises, has long understood the benefits of the open-source model. The active community of developers contributes to security, reliability and adherence of the product to the latest standards. Starting from GNU OS and Linux, through PHP and continuing to Ruby on Rails, Hadoop, Memcached and Redis, the open-source software was possibly one of the driving forces to create the today's landscape of successful software mega-enterprises [1].

Inspired by the open-source software model success, the Silicon IP reuse paradigm has emerged. Open-source hardware enables reusing hardware components and by this shortens the design cycle of systems on chip. Some examples of the early open-source IC initiatives are the Opencores project [2], OpenPOWER [3], OpenRISC [4] and OpenSPARC [5]. The recent introduction of the RISC-V architecture [6] have produced a big open-source bang both in the academic and in the industrial communities. Enthusiasm about the open source IC code lead to the establishment of several RISC-V-centric open-source SoC projects, such as OpenTitan [7] and others [8], [9], [10]. Even though the RISC-V creators intended it to be only an open-architecture project, the RISC-V community is now the main driver of the open source hardware IP.

Arguably, product security is one among many beneficiaries of the open-source hardware model. In line with the Kerckhoff's law, the white-box model increases the confidence in the contents of the IC [11]. Admittedly, security of the open-source hardware can be a controversial topic. The open-source model can be both good and bad news for security [12], and there are publications that expose its threats [13]. The answer depends on the application and the policy. This article addresses the wide community that has already chosen open source as a contributing factor to security. In any case, open source alone is hardly sufficient [14]. In contrast to software, authenticity of which is easy to verify, for integrated circuits that go a long way from the source to the manufactured and packaged device, validating the source of a given unit is far from being an easy task. The IC supply chain involves multiple parties with various levels of trust, so that integrity can be compromised along the chain either by a malicious or by a dishonest party. Vast research has been devoted to the threats of the IC supply chain, such as hardware Trojans [15], [16]. A malicious IC vendor may claim adherence to the open-source code, but modify it at his discretion to insert a Trojan. A malicious service provider may modify the contents to inject a Trojan during one of the stages along the chain, such as synthesis, place and route, GDSII generation and mask fabrication. In additions to Trojans, there are economic incentives. Consider for example a dishonest SoC designer that aims to save costs by weakening the countermeasures against side-channel attacks. In this case, the product will have the same functionality, but its security will be compromised.

Addressing these threats requires either establishing trust in the entire supply chain, which is usually impractical, or finding a way to validate that the manufactured product is based solely on the declared source. This resonates with the Trojan detection problem, for which many approaches were proposed, but so far no universal practical solution exists. In contrast to the general Trojan problem, here the validator has a clear advantage of the available golden model and a certification program that may enforce specific implementation.

To correlate between the source and the finished product, one can examine the chip development flow and apply a transitive validation approach by comparing between an input and an output of each stage. Figure 1 shows a simplified chip development flow. The way from the Register Transfer Language (RTL) source to the manufactured device comprises several stages that start from software representations and proceed to physical manufacturing steps. Automatic verification tools such as formal logic equivalence and Layout-versus-Schematic (LVS) can cover software representations. The problem arises in the stages that deal with tangible material: stages like mask fabrication, wafer manufacturing and packaging. To validate them, one needs to do reverse engineering of the physical outcome of the specific stage and compare it to the input to this stage. For example, the wafer manufacturing stage gets the mask set at the input and produces a multi-layer Silicon wafer. Verifying the equivalence between the input and the output of this stage involves checking that each produced layer matches its corresponding mask. Although IC reverse engineering tools that acquire layer images do exist, they are expensive and not sufficiently accurate. Even if such an investment of time and money can be afforded, and a single part can be verified for equivalence, there is no guarantee of the authenticity of the other parts. Moreover, if there is no trust in the service provider, there is no point in checking separate stages, since there is no guarantee that the

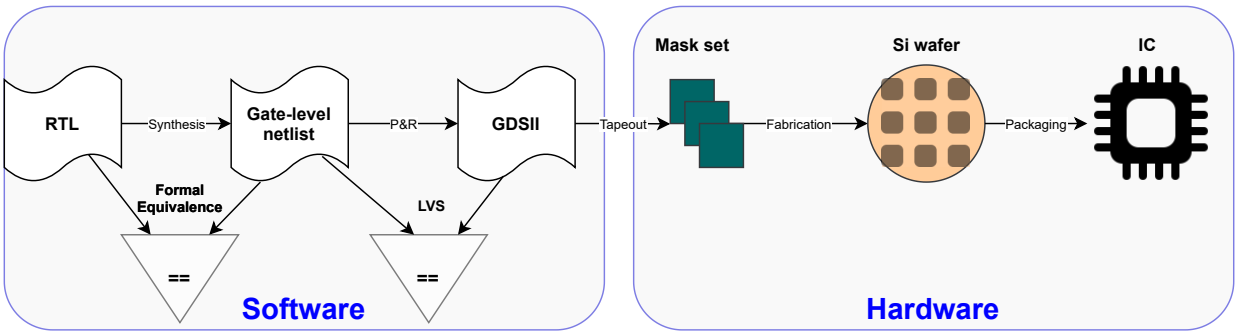

Fig. 1: IC development process and transitive verification possibilities.

provider used the input to the stage that he claims to use. Thereby, the only reliable method is an end-to-end verification, namely a process that verifies the logic equivalence between the final product and the source. This process should be cost-effective, fast and built to apply to a large population of parts. The latter also means that it must be non-destructive.

In this paper, we present the Open Scan methodology, a new step that we propose to add to the open source certification program, in particular of the cryptographic devices. The IC vendor will be required to expose the production scan harness of the open-source based hardware, and the certifying body with the IC customer may use it to check the conformance of the IC to the original model. Open Scan builds on the production scan mechanism embedded in virtually every digital integrated circuit, therefore it consumes no extra resources. Production scan allows for the automatic test pattern generation (ATPG) used by the production sort program to identify damaged parts to be discarded. Scan provides serial access to all the internal registers from the interface of the IC. The production tester can use this access channel to examine the digital logic in the IC and to verify that it behaves as expected by comparing the values sampled in the registers to the results obtained from the simulation model. We propose to use the same approach to compare the IC contents to the model available as open source. This comparison comprises several stages, which include register correspondence generation, register dependency graph comparison, combinational logic equivalence check and search for hidden state. The logic equivalence check of IC vs. the reference model is a hard task due to the lack of structural information from the IC. We propose to use an ATPG-based approach as a heuristic solution to the equivalence check problem.

Open Scan is intended only for fully open-source IC projects. It addresses a specific problem of genuineness of the open-source RTL-based cryptographic products, and it does not replace generic hardware Trojan detection methods. Open Scan is effective against several classes of source modification attacks. The first class is downgraded secure functionality. The second class is malicious modifications (Trojan horses) inserted by a party (such as the manufacturing facility) that has limited modification capabilities, which prevent it from inserting a sophisticated Trojan hidden from scan. The third class is malicious modifications that can be hidden from scan. Although Open Scan as is cannot address those attacks, we suggest a methodology that significantly hampers the attacker's ability to inject such modifications. The methodology is based on comparison between the scan and mission modes.

II. Background

A. Scan Technique and Reverse Engineering with Scan

Scan insertion is a widely used Design-For-Test (DFT) technique that allows for the automatic generation of test vectors for production test of integrated circuits. Thanks to its efficiency and ability to achieve high coverage, it has become a de-facto standard for testing digital circuits, supported by all the major synthesis tools. Moreover, scan insertion is usually enforced by ASIC vendors.

The scan insertion algorithm introduces a new operational mode, $shift\ mode$, which arranges the internal registers in one or more shift registers, called scan chains . The production tester may switch the chip to the shift mode and use the scan chains both to place the chip in the desired state ($ShiftIn(vec_{in})$) and to sample its current state ($vec_{out} = ShiftOut$). The $ShiftIn$ and $ShiftOut$ operations can be combined with a single ($Capture$) clock cycle in the mission mode to construct a sequence that sets the state of the IC and tests the response of the cumulative combinational function F to this state. We denote this as a probe operation ($vec_{out} = Probe_F(vec_{in})$), comprising three-operation sequence: $ShiftIn(vec_{in}) \rightarrow Capture \rightarrow ShiftOut$. This effectively turns the testing problem from stateful to stateless, namely combinational.

The scan testing aims to cover all possible faults in the IC, where the fault stands for an effect of a production defect on the functionality of the IC. Based on the classical 'stuck-at' model, two faults are assigned to every net: 'stuck-at-0' and 'stuck-at-1'. A production test is said to cover a fault when the faulty unit produces response on the test stimulus different from the response of the good part. Using the scan infrastructure, the automatic test pattern generation (ATPG) tools create test vectors that produce maximum coverage of all the faults [17] under constraints. Two conditions have to hold for a fault to be covered: (1) the net corresponding to the fault must be driven with a value opposite to the fault value (fault sensitization); and (2) the value of the net has to be propagated to an output (fault propagation).

In addition to being an effective DFT technique, scan provides a convenient channel for reverse engineering of the digital part of the integrated circuit. Thanks to scan, the learning problem, analogously to the testing problem, turns to combinational. Therefore, performing an exhaustive search over all possible states of the device's registers and input pins will reveal a truth table that fully describes the function F. The exhaustive search itself is exponential in nature, and therefore not feasible. However, the function F can be approximated using Boolean function and other learning algorithms [18]. This approximation can be good enough for many practical purposes [19].

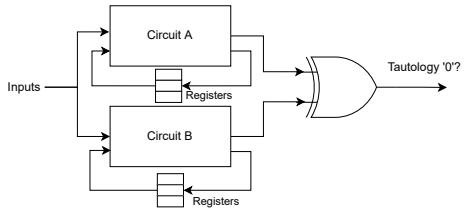

Fig. 2: The miter construct used to check logic equivalence between single-output circuits A and B presented as finite state machines. If the circuits are equivalent, the XOR output is constant '0'.

B. Logic equivalence checking

Logic equivalence checking is used in the IC development cycle to compare between different views of the design, for example between the RTL entry and the gate-level netlist or between the original netlist and the one extracted from the post-place-and-route data.

Often, when checking equivalence, both designs are regarded as a Finite State Machine (FSM). A product machine of the two circuits can be formed by connecting each corresponding primary input pair of the two machines together, and connecting each corresponding primary output pair to an XOR gate. This model is also referred to as the miter [20]. Figure 2 illustrates the miter for two logic circuits A and B. If A and B are logically equivalent, the XOR gate connected to the circuits outputs returns 0 for any combination of values applied to the miter's inputs for any reachable state. Therefore, the tools assume the same state encoding for both circuits. This assumption is justified in the classical applications of the equivalence checking, because the IC implementation tools, such as the synthesis tools, retain most of the sequential elements. Some synthesis tools may perform local sequential optimizations, removing unnecessary flip-flops and optimizing state machines. These special cases are handled by using local sequential equivalence checks.

Given the state encoding equivalence and known correspondence of the state registers between the two circuits, the check task is reduced to combinational. The registers are removed from the miter, and the primary inputs are joined with the flip-flop outputs. In addition, a new miter is created for each flip-flop input.

Even after the reduction to combinational, the task still remains NP-hard. While circuits with a small number of inputs can be exhaustively simulated for all possible input vectors, for larger circuits this brute-force method is impractical. In this case, an incremental verification is employed [21] with the general idea of optimizing the miter to bring the complexity down to the acceptable level. This can be done when the circuits exhibit some similarity, for example if they have common signal names or hierarchical structures.

In [22], Agrawal makes an interesting observation that comparing two circuits using a test set that covers all possible stuck-at faults in both circuits, produces correct answer with high probability. To get the intuition behind this statement, observe that the combination of ATPG test patterns that cover all the faults in circuits A and B in Figure 2 also covers the stuck-at-0 fault at the output of the XOR gate.

III. Open Scan Methodology

A. System, Roles and Infrastructure

The ecosystem we use to demonstrate the open scan methodology comprises the following actors:

a) Open source organization: a roof organization that manages the open source data, defines laws and regulates the development and usage.

b) IC vendor: a company that develops and markets a product based on the open source.

c) Subcontractors: organizations that provide service to the IC vendor at different stages of product development and production.

d) System integrator: a company that purchases the product and integrates it into the target application.

e) Validator: a body that validates the conformance of the product to the source. The validator can report either to the open source organization or to the system integrator.

The open scan methodology works in the transparent open source model, in which the IC vendor is required to update the repository with all the changes made to the source, and all the digital parts of the IC are based only on open source. The rules are regulated by the open source organization. The vendor will be also required to provide information about the scan operation protocol, which includes the procedure to place the target device into the scan mode, instructions about accessing the scan chains, scan architecture description, scan chain order, and in general everything the validator needs to apply scan vectors. The IC vendor is also expected to provide the register correspondence mapping, which associates each register on the scan chain with the corresponding RTL register. In addition, the IC vendor will provide maximum-coverage ATPG vectors that will be used for equivalence checking (Section III-E). We denote the information above as scan metadata.

In general, it should be in the vendor's interests to provide the tools for proving the conformance to the open source model. In addition, such a cooperation may be formulated as a requirement for conformance to the relevant open-source certification program. In that case, the certification body will need to establish rules for Open Scan, when the rationale is providing the validator with the means to run the equivalence check.

The IC should be placed in a receptacle to run scan. For that purpose, the IC vendor may provide a test board or other type of an embedded system. For example, the load board used to host the device under test, when it is placed on the production tester is a good fit. Since such systems may incur high costs, it may suffice to require only the reference design, so that the validator will build the board on his own.

The IC vendor can be either honest or dishonest. An honest vendor provides accurate information on the scan structure and operation, in which case the potential adversary can be one of the subcontractors. In the honest vendor model, the vendor's information is trusted, but the IC content is untrusted. A dishonest IC vendor may falsify scan metadata to obfuscate the deviation of the IC from the original source. In this section, we lay out the verification approach for the honest vendor model. In Section IV-A, we extend the discussion to the dishonest vendor model.

B. Verification Principle

As in Section II-B, we treat the IC logic as a FSM. We start from verifying the equivalence of the FSM state register to RTL, namely the mapping of the RTL registers to the locations on the scan chains. This information is provided by the IC vendor (Section III-A), albeit is not necessarily trusted (Section IV-A). The following step covers the combinational logic, which is derived using scan. This section outlines the method of learning the combinational logic from scan.

The verification starts from validating the register correspondence, provided by the IC vendor, namely checking a one-to-one correspon-

dence between the scan cell set and the set of sequential elements in RTL. This allows for running the combinational logic equivalence verification (Section II-B).

Initially, the combinational logic can be represented by a black-box oracle based on the $Probe$ operation (Section II-A) that receives a vector of the chip state, stored in the scan registers, and returns the next state, namely the scan registers values after running a single scan capture cycle. Furthermore, the vector oracle can be split into a collection of bit oracles. In the digital circuit domain, the bit oracle corresponds to the *combinational logic cone* (Definition III.1), a subcircuit comprising all the combinational logic gates in the transitive fan-in of a sequential element (a register).

The IC design must comply with a set of rules (scan design rules) to be scan-ready [23]. For example, it must be fully synchronous and edge-triggered, and clock inputs of the internal registers must be controllable from the IC interface. Typical modern designs comply with most of the rules, and the parts that don't comply are modified when the device is placed in the scan mode. For example, an internally generated clock is replaced by an external clock. As a result, the IC functionality in the scan mode deviates from its functionality in the mission mode. This deviation, though, must be small enough to prevent notable impact on the test coverage. For the sake of further discussion, we assume that both the IC and the model are compared when placed in the scan mode. In Section IV-A, we discuss the pitfalls of this assumption.

The scan design rules ensure that the logic subject to scan test can be represented as a simple automaton, whereas the scan registers represent the state, and the combinational logic represents the next-state function. Furthermore, it is possible to derive the next-state function for each register from the functionality of a subcircuit comprising all the combinational logic gates in the transitive fan-in of this register, i.e. a logic cone. Let us now formalize the scan-based equivalence check principle.

Definition III.1. *Given a circuit S comprising a set of sequential gates (registers) R and a set of combinational gates A, for every register r_i, a* ***logic cone*** *A_i is a subset of A, which includes all the elements in A, from which a combinational path exists to some synchronous input of r_i.*

The definition above implies that the logic cone is a combinational subcircuit of S, such that its inputs are outputs of registers, and all of its outputs are inputs to one register. For simplicity, we assume that all the registers are D flip-flops. Other types of flip-flops that have additional synchronous inputs can be converted to a D flip-flop with combinational logic. As a result of this conversion, each logic cone will have one output only.

Lemma III.2. *Let S be a fully-synchronous digital circuit comprising (1) n sequential elements (registers) triggered by a single clock, and (2) combinational gates connected to the sequential elements and between themselves. The sequential elements comprise a state register $r = (r_1, \ldots, r_n) \in \{0,1\}^n$, and the combinational logic implements the next state function $F(r) = (F_0, \ldots, F_n)$. Let S_{ref} be a reference RTL representation of a fully-synchronous digital circuit with a n-bit wide state register r^{ref}, such that the state correspondence is known a priori (for every $i = 1, \ldots, n$, r_i corresponds to r_i^{ref}). S is logically equivalent to S_{ref} if: (1) r and r^{ref} have the same initial value and (2) for every value v, $F(v) = F^{ref}(v)$.*

The conditions above are sufficient, but not necessary. In a general case, two state machines are equivalent if for any given sequence of inputs, both produce the same sequence of outputs. However, we do allow ourselves to assume that the IC netlist was generated from RTL using automatic synthesis tools, which preserve the registers functionality with some exceptions, discussed in Section IV-B2. Therefore, we will require equivalence not only for the sequence of outputs, but also for the sequence of states. This effectively turns the conditions to necessary. Here, we also ignore the primary inputs and outputs without loss of generality. The a priori register correspondence is required to avoid isomorphism-based equality.

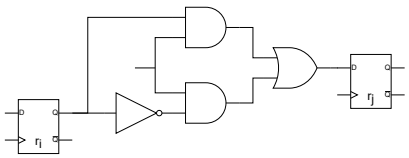

Fig. 3: False dependency of register r_j on register r_i.

The next-state function F can be split into n single-bit functions F_i, thus the comparison can be performed for each bit individually. Notably, F_i is logically equivalent to the corresponding logic cone. There are many ways to map the Boolean function F_i to a logic gate-level circuit. Since our goal is logic equivalence, the specific mapping is irrelevant.

The logic cone is a combinational circuit with multiple inputs and one output. Definition III.1 implies a combinational path between any input of the cone and its respective register r_i and the cone's output and its respective register r_j. Such a combinational path must exist if there is a combinational logic dependency of r_j on r_i. We denote it as a dependency link. We can then represent these links as edges in a directed graph with vertices designating registers. This is a register dependency graph of the circuit S. Notably, a combinational path from r_i to r_j does not necessarily imply logic dependency. We denote such paths as false dependencies (Figure 3), which are unlikely in real designs.

To quantify the dependency, we use $Influence$, a measure of the dependency that designates the extent to which certain input affects the function [24]. Namely, the influence of the variable x_i on function f: $\{0,1\}^n \to \{0,1\}$ is a probability that for a random input x, inverting the variable x_i changes the output of the function.

$$Inf_i[f] = \underset{x \sim [0,1]^n}{Pr}[f(x_1, \ldots, x_i, \ldots, x_n) \neq f(x_1, \ldots, \neg x_i, \ldots, x_n)] \tag{1}$$

Definition III.3. *For two registers r_i and r_j in a fully synchronous circuit S, a dependency link $l_{i,j}$ exists if there is a combinational path from the output of r_i to the input to r_j and $Inf_{r_j}[r_i] > 0$.*

Definition III.4. *A register dependency graph $\mathbb{G}_S = (V_S, E_S)$ of a fully synchronous circuit S is a directed graph with vertices designating the registers of S and edges $(i,j) \in E_S \iff \exists l_{i,j}$ designating the dependency links between the corresponding registers.*

Equivalence of two circuits implies isomorphism of their respective dependency graphs. In the presence of the register correspondence list as required in Section III-A, the graph vertices can be labeled, reducing isomorphism to equality.

Lemma III.5. *Two fully synchronous circuits S_1 , S_2 with full state bit correspondence are logically equivalent at the state sequence level only if their respective register dependency graphs are equal.*

Thus, a necessary condition for the (state sequence-based) equivalence of two circuits is the equivalence of their register dependency

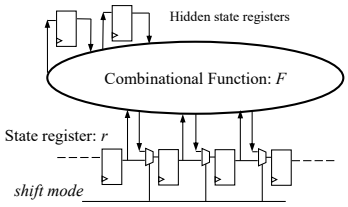

Fig. 4: Hidden State. The registers on the top are not connected to the scan chain, hence cannot be directly controlled by scan.

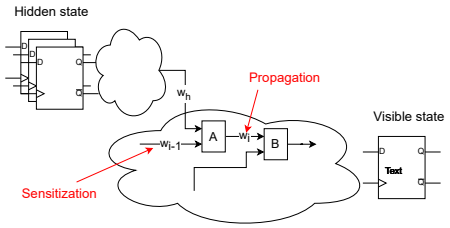

Fig. 5: Hidden state merging with visible logic via gate A. The ATPG vector set should sensitize w_i and propagate w_{i-1}.

graphs. This in turn implies equal number of registers. A sufficient condition for the equivalence is formulated in Lemma III.2 as an equivalence of the function $F(v)$ for every v. The proof of this equivalence is an exponential task with the number of registers. However, given the dependency graph, the complexity can be reduced by comparing each logic cone individually.

Drawing from these conditions, we can build the verification flow as a multi-step process, checking all the necessary conditions, starting from the easiest and gaining confidence with each step. The Open Scan verification flow starts with register correspondence (Section III-B), followed by dependency graph comparison (Section III-C), search for hidden state (Section III-D) and ATPG-based logic cones equivalence check (Section III-E).

C. Dependency graph extraction

The objective is verifying equivalency of the register dependency graph $G_{impl} = (V_{impl}, E_{impl})$ of the implemented device to the graph $G_{rtl} = (V_{rtl}, E_{rtl})$ of the reference RTL representation S_{ref}. Extraction of G_{rtl} is straight-forward. For this, we map the RTL to a generic logic library, containing sequential and combinational elements. The dependency graph is derived from traversing the logic paths in the mapped RTL. Variety of synthesis tools, as well as specialized tools for reverse engineering [25] can be used for that purpose. Since this graph is built on structural and not logical connections, it may contain false dependency links.

In contrast to the RTL reference, no a priori structural information exists for the implemented device, and the only tool in our possession is scan. Scan queries can be used to construct a partial dependency graph [18]. The construction algorithm's complexity in $O(2^k)$, where k is a parameter of the algorithm. The algorithm finds with high probability all the dependencies with the influence $Inf > 2^{-k}$.

In practice, the algorithm can find the majority of dependencies in reasonable time [26]. The resulting register dependency graph $G_{scan} = (V_{scan}, E_{scan})$ is a subgraph of G_{impl}. If the design implementation is equal to the RTL reference, G_{scan} will contain the same set of vertices and a subset of edges of G_{rtl}. Namely, we will require $V_{scan} = V_{rtl}$ and $E_{scan} \in E_{rtl}$.

To prove strict equivalence between the RTL and the implementation dependency graphs, two more conditions should hold:

1) All the non-false dependency edges from E_{rtl} that do not appear in E_{scan} do exist in E_{impl}.
2) All the edges that exist in E_{impl} also exist in E_{rtl}.

The first condition can be verified in the following way. For every edge $E_{i,j}$ from $E_{rtl} - E_{scan}$, find a distinguishing vector $\bar{x}$ for input index i and output index j such that $f_j(x_1, \ldots, x_i, \ldots, x_n) \neq f_j(x_1, \ldots, \neg x_i, \ldots, x_n)$. This is exactly what the ATPG algorithms do for fault propagation. Hence, ATPG algorithms can be used to find the distinguishing vector. This vector can then be applied using scan first with $x_i = 0$ and then with $x_i = 1$. If the two applications produce different f_j, then the edge $E_{i,j}$ exists in G_{impl}. Failure of the ATPG algorithm to find a distinguishing vector indicates with high probability that the edge corresponds to a false dependency. In such a case, this edge will be removed from G_{rtl}, and the comparison will be reapplied. Naturally, the dependency graph extracted from scan does not contain false dependencies.

To sum up, following are the steps of checking equivalence of the register dependency graphs between the device implementation and its respective RTL reference.

1) Extract RTL reference register dependency graph $G_{rtl} = (V_{rtl}, E_{rtl})$ using design exploration tools.
2) Extract partial design implementation register dependency graph $G_{scan} = (V_{scan}, E_{scan})$ using scan.
3) Verify $V_{scan} = V_{rtl}$ and $E_{scan} \in E_{rtl}$.
4) Verify that all the edges in $E_{rtl} - E_{scan}$ exist in the design implementation using distinguishing vectors removing all the probably false edges (which no vector was found) from E_{rtl}.

D. Hidden state

We draw our methodology from the fact that scan essentially converts the device to a combinational function (Section II-A), provided all the registers connected to the scan chain(s). If this condition is not met, a *hidden state* is created. Whether created intentionally or by mistake, it substantially raises the verification complexity. Hence, the open source organization shall demand full scan coverage of all the registers in the qualifying device. The validator has to enforce this requirement by detecting the presence of a hidden state. Note that if a hidden state exists, it will necessarily contain registers, which are not part of the published source.

We argue that harmful hidden states must both depend on and affect the visible state. Namely, we assume a mutual dependency between the state r and the hidden state r^h (Figure 4). Lack of dependency of r on r^h effectively means that r^h belongs to a logically disjoint circuit and, excluding scenarios where a Trojan has its own communication means like RF antennae, can make no harm. Likewise, if there is no dependency of r^h on r, the hidden circuit does not have read access to the secrets, hence its capabilities are very limited.

Intuitively, to detect the hidden state using scan, first we need to find a value of the visible state, for which the hidden state value affects the next state of r. We call it a distinguishing value. In a more formal language, the hidden state r^h is in the domain of the next state function $r^{next} = F(r, r^h)$, which implies existence of at least one distinguishing value r^d, s.t. $\forall r^{h0}, \exists r^{h0n}, s.t.\ F(r^d, r^{h0}) \neq F(r^d, r^{h0n})$. Namely, if the state register contains the value r^d, the value of the hidden state register affects the next state. If the hidden scan state depends on n_k bits from r, there will be at least 2^{n-n_k} distinguishing values of r, and the probability to find at least

one using brute force will be $\geq 2^{-n_k}$. Assuming without loss of generality that $F_{ref}(r^d) = F(r^d, r^{h0n})$, that is the next state for r^{h0n} is equal to the next state as if there was no hidden state, we need to find the pair of values r^d, r^{h0}, for which the next state will be different from next state in the reference model for r^d.

Algorithm 1 is a random heuristic algorithm essentially performing a random search for a tuple (r^d, r^{h0}). The algorithm searches for a distinguishing value among the vectors from the ATPG set, assuming that these vectors reflect the functionality in the best way (see the reasoning further in this section). In addition, to increase the probability of finding r^h, the algorithm performs sequential ATPG by executing several capture cycles, assuming that the hidden state may contain an automaton that starts from an "interesting" functional state, which at the end modifies the main state. After each capture cycle, the algorithm compares the result to the reference model.

Algorithm 1 Hidden state search ($MaxDepth$ – algorithm parameter)

```
for vec_in in ATPG Vector Set do
    TestVectors = {}
    for depth = 1 to MaxDepth do
        ShiftIn(vec_in)
        Perform depth Capture cycles
        vec_out = ShiftOut
        append vec_out to TestVectors
    end for
    for testvec_in in TestVectors do
        testvec_out = Probe_ref(testvec_in)
        if testvec_out ≠ next item in TestVectors then
            Halt: hidden state found
        end if
    end for
end for
```

Let us demonstrate the rationale of this algorithm in two steps:

a) Step 1: Assume that the hidden state depends only on the visible state and not on itself, that is $r^{h,next} = F_h(r)$. Then, one capture cycle is sufficient to set the hidden state to any desired value. With probability of at least 2^{-n_k}, the visible state r^{next} after the capture will be a distinguishing vector. Two consecutive capture cycles will give $r^{next2} = F(r^{next}, r^{h,next})$. In the absence of the hidden state, performing a single capture after shifting in the vector r^{next} should result in r^{next2}. Hence, we can test this vector in simulation on the reference model, and a different answer than r^{next2} will prove the existence of the hidden state. Note that for the case of $r^{h,next} = F_h(r)$, only one additional capture cycle is required, thus $MaxDepth = 2$ should be used. For success, we need at least one distinguishing vector r, for which $F(r^{next}, r^{h,next}) \neq F_{ref}(r^{next})$. The worst case probability to find the hidden state using this approach will be $Prob = 1 - (1 - 2^{-n_k - |r^h|})^{NV}$, where NV is the number of tested vectors.We expect a significantly higher probability in a typical case. In particular, the ATPG vector set will likely contain a distinguishing vector. A simplified example in Figure 5 illustrates it. Let's assume that the hidden-state based logic affects the visible state via a single (taking the worst case) wire w_h, which merges with the rest of logic via gate A. The input w_{i-1} connected to an input of A should be sensitized and a wire w_i connected to the output of this gate should be propagated by the ATPG vector set. Hence, to make a fault on w_h propagate to a scan flip-flop, what's left is make it propagate through gate A, which can be achieved with high probability provided sufficient number of vectors and assuming a small number of inputs to A.

b) Step 2: Now we remove the constraint on the hidden state from Step 1, so that $r^{h,next}$ can depend on r and on itself. That means $MaxDepth = 2$ may be insufficient. In this case, we add more capture cycles to try and cover as many hidden states as possible. If one of the values of r along the way of the capture sequence is a distinguishing vector, checking the same vector on the reference model may reveal the presence of the hidden state. Hence, all the vectors along the capture sequence are recorded and then applied to the reference model. Since here the hidden state implements a state machine that does not provide shortcuts, the worst case depth is equal to the number of possible hidden states, equal to $2^{|r^h|}$.

E. Logic Equivalence Verification

The state register correspondence and the dependency graphs obtained in the previous stages allow for running a formal equivalence check between RTL and IC one logic cone at a time. The heuristic equivalence check algorithms (Section II-B) exploit metadata, such as signal names, as well as a structural similarity of the two circuits to reduce complexity. None of these is available in our case, since we don't have a circuit description of the IC. Instead, the IC logic cones are represented by oracles providing a vector answer on a vector query. Hence, no internal structure-assisted verification techniques can apply. However, Agrawal's [22] heuristic relies on the structural information just to create the ATPG vectors. The verification that follows is merely a simulation of the entire miter as a black box, for which the oracle suffices. Thanks to the rules that enforce publishing the maximum-coverage ATPG vectors (Section III-A), the ATPG vector set for the IC is available from the IC vendor. Hence, this method fits the needs of Open Scan. Although relying on the vendor data for verification may contradict the goal of the verification , our threat model does not necessarily mark the vendor as dishonest. Additional entities are involved in the process. Moreover, forging ATPG vectors in a way that they still pass on IC is a challenging task, and this won't necessarily help, since the equivalence verification also includes the ATPG vectors for the reference model. We discuss this in more details in Section IV-A.

The Agrawal's method combines ATPG vectors for both circuits, so that vectors for the RTL reference have to be created. Since ATPG can only run on a gate-level netlist, the reference RTL should be first synthesized. Clearly, there is a transitive relation: for gate-level netlists G_1, G_2 and a reference model M: $G_1 \equiv G_2 \cap G_1 \equiv M \implies G_2 \equiv M$. Namely, it should be sufficient to check the equivalence between any mapping of the reference model to gates and the IC scan-based oracle. The mapping can use also a generic technology-independent library of gates as long as it allows to run ATPG on it.

The following sequence summarizes the proposed equivalence verification:

1) Map the RTL reference model to a gate-level netlist
2) Obtain input vector set T_{ref} by running ATPG on the RTL reference model mapped to netlist
3) Combine T_{ref} with the IC vendor's ATPG vector set T_{IC}
4) Verify ATPG test with T_{IC} passes on IC
5) Apply the combined vector set T_{comb} to the IC using scan and to the reference model using simulation and verify they provide matching results.

IV. Discussion

Section III assumes an honest vendor model; additionally, it makes several assumptions about the design and the implementation flow. Moreover, the algorithms have practical limitations, but we make statistical assumptions, under which they find the deviations in most of the cases. Below, we discuss the justifications of the model assumptions and draft possible actions for when they are not met.

A. The dishonest vendor model

A dishonest IC vendor provides misleading information to the validator to prevent him from finding the deviation of the implementation from the claimed open-source model. This misinformation can be injected in several places as discussed below.

a) Forged register correspondence data: The vendor is expected to give correspondence data for all the registers that exist in RTL, hence no RTL registers could be missing, and the scan chains lengths are easily verified. Adding unreported scan chains essentially creates a hidden state, addressed in Section III-D.

b) Forged ATPG vector set: Here, need to keep in mind that the vector set must still pass testing on IC. Therefore, the forging capabilities are limited. The vendor can still provide vectors with partial coverage by excluding the vectors that expose the deviation. However, there is a substantial chance of the deviation to be detected by the vectors generated for the reference model. Our experiments show that almost all the modified benchmarks were categorized as modified even when using only the ATPG set of the reference model.

c) Logic obfuscation in scan mode: The IC vendor may present different logic in scan mode than in mission mode. As an extreme example, consider a completely-simulated scan, i.e. the ATPG response is just read from an internal ROM. Although, this extreme case can be easily detected by adding more vectors. However, a small Trojan circuit can be completely hidden. This is arguably the most serious limitation of the open-scan verification framework when applied to malicious modification by the IC vendor himself. Together with that, this obfuscation is not effective for the economically motivated dishonest vendor, since the logic that was intended for saving will be added back in scan mode.

Open Scan can be enhanced for better detection of obfuscated Trojans. Since the obfuscation counts on access to the indication of a 'scan mode', the idea is to verify that the 'scan mode' logic is equal to the mission mode logic without exposing the register contents in mission mode. This can be achieved by adding a light-weight hash function that will generate a signature of the register contents in mission mode.

d) Good and bad ICs.: A malicious actor can manufacture two revisions of the product: a good revision for testing and a bad revision, containing a malicious circuit for field. To counter this, the validator can perform periodic testing of randomly selected parts. For security critical applications, each part can be tested.

B. Mapping ambiguities

The algorithm in Section III implies that the RTL reference describes a single logic circuit. In fact, there is even stronger assumption – these two mappings have to be logically equivalent at the resolution of the logic cones. This assumption is not necessarily met in all the cases. Below, we discuss two typical cases.

1) Don't care conditions: RTL allows to describe logic with don't care conditions. It is commonly used, for example when describing state transition of a FSM, in which only only part of the state vector space is used for the state encoding, while the remaining part is unused. Don't care conditions automatically imply more than one logic implementation.

The don't care conditions may lead only to false negative verification results. Empirically, we didn't observe failures resulting from the don't care conditions, for example when comparing two gate-level netlists, generated with different synthesis tools (Section V-A). This may mean either that both tools map the don't care conditions identically or that the don't care terms do not generate new faults, and therefore not chosen by the ATPG algorithms. The empirical evidence suggests that the number of false negatives caused by the don't care conditions is low. Therefore, each case can be handled manually by checking whether the distinguishing vector contains a don't care condition for the failing output. Automatic checks are also available. Such checks are based on three-value similar algebra [27]. The existing methods are built for the typical usage of comparison between RTL and netlist, so they can take advantage of the netlist structure. They can be adjusted to fit our usage model. It should also be mentioned that design for security discourages using don't care conditions. For example, reaching an unassigned FSM state encoding should be treated as a tampering attempt.

2) Sequential transformations: So far we assumed full state register correspondence in a sense of equivalent functionality of registers in the reference and the implemented circuits. In other words, the synthesis and other tools are assumed to modify only combinational logic and leave the sequential elements intact. However, the synthesis tools may perform local sequential optimizations, such as retiming and sequential redundancy removal.

In general, we can assume that all the sequential transformations that are handled by the commercial logic equivalence check tools, can also be applied to the Open Scan verification flow.

C. Preventing attacks via Open Scan

Scan can simplify not only reverse engineering, which is not a threat for open-source designs, but also attacks on confidentiality and integrity of the application itself [28], [29]. Access to the internal registers and logic can expose application confidential data as well as compromise application integrity. Making scan metadata public, as required by Open Scan, facilitates the attacker's needs even more. Although this seems as a security weakness introduced by Open Scan, several solutions exist that can eliminate its effect [30], most of which are already in use. Below are two examples:

a) Enforcing reset when switching between modes.: Access to the volatile internal state relies on the ability to switch dynamically from mission mode to scan mode and back to retrieve or alter the real-time information. To prevent this, the IC enforces full reset when switching between the modes [31].

b) Scan lock after verification.: The device life cycle can define a separate phase for Open Scan verification, where scan access is open. After this phase, for example during provisioning, scan mode can be permanently disabled or one of the known authentication mechanisms can be switched on.

V. Experimental results

A. Benchmarks and tools

All the experiments were performed on simulated models. We used benchmarks from the Trust-Hub Trojan database [32], as well as the following open-source RTL designs: a toy RISC-V model [33], IBEX RISC-V [34], tiny AES [35] and SHA-256 [36]. The approximate standard cell count figures of the designs are $6K, 14K, 145K$ and $180K$ respectively. Synopsys Design Compiler and Cadence Genus were used to synthesize the RTL sources, insert scan and generate the

TABLE I: Trust-hub results

Benchmark	Hidden state	Stage
AES-T2300/2400	No	ATPG
AES-* except T2300/2400	Yes	Prep (diff chain length)[1]
B15-*	Yes	Hidden state search
B19-*	Yes	Hidden state search
BASICRSA-T100	No	Failed to detect[2]
BASICRSA-T300/400	Yes	Hidden state search
EthernetMAC-T100-T600	No	Failed to detect[3]
EthernetMAC-T7*	Yes	Hidden state search
MC8051	No	ATPG

[1] Trojan flops were connected to scan
[2] Detectable with the implementation ATPG
[3] Timing Trojans

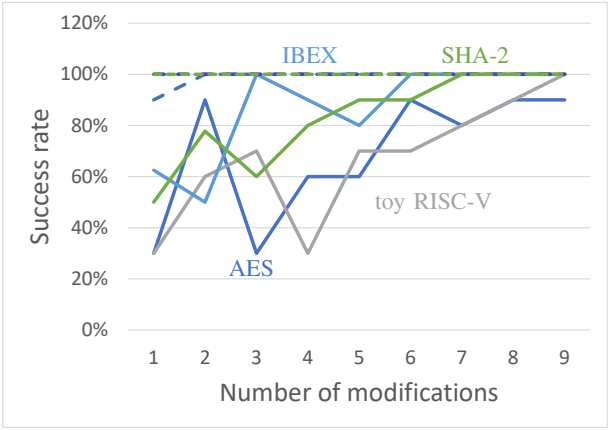

Fig. 6: Success rate of the logic deviation detection applied to four benchmarks. Solid lines indicate dependency graph comparison and dashed lines – ATPG-based checks.

netlists. The synthesis tool reports were used to obtain the register correspondence. Synopsys TetraMax was used to create the ATPG vectors. HAL, the hardware analyzer framework [37] was used to insert modifications to the netlists and to run dependency analysis. Cadence Xcelium was used for the ATPG simulation.

B. Results

a) Proof of concept: For each benchmark, we used two different synthesis and scan insertion flows from different EDA vendors. After obtaining two implementations, we applied the ATPG stimulus created by one of the flow to both (adjusting the scan chain order) and verified it passed on both netlists. This provided evidence of the invariance of the scan logic to the scan insertion method.

b) Trust-Hub benchmarks: We started the evaluation from checking how the Open Scan flow detects Trojans from the Trust-Hub Trojan database [32]. The Trojan-free circuit played the role of the reference model, and the Trojan-inserted circuit – implementation. The benchmarks provided in a netlist form were used as is, and RTL-based benchmarks were synthesized. We tested all the benchmarks from the database except benchmarks having synthesis or DFT rules compliance issues and netlist benchmarks without scan metadata.

The flip-flop dependency graph of the golden model was created using HAL by traversing netlist connections. This graph contains all combinational dependencies by construction. For the deviated netlists, the dependency graphs were created in a similar way, however, dependencies with Boolean influence lower than a threshold of $\frac{1}{30,000}$ were excluded from the graph to emulate the limitations of the scan-based extraction. According to the procedure defined in Section III-C, first we verify $E_{scan} \in E_{rtl}$, and if this relation holds, we generate the distinguishing patterns from ATPG and check whether the dependencies corresponding to the remaining edges in E_{rtl} exist also in the implementation.

Although in Section III-E, we require an ATPG vector set combining the vector sets T_{ref} from the reference model and T_{IC} from the vendor, here we assume a worst case, namely a dishonest vendor, and therefore the only reliable vector set is T_{ref}. Hence, we generate ATPG vectors only for the original benchmark netlist, based on which the logic equivalence check is performed.

Table I shows the evaluation results with the Trust-hub benchmarks, grouped by the original circuit type. Some of the benchmarks were provided in a netlist form with scan. However, Trojan was not part of scan, which created a hidden state. This hidden state was easily detected using our methodology. As can be seen from the table, two Trojans types evaded detection by the Openscan flow. BasicRSA-T100 contained a low-influence Trojan that was not covered by the reference model's ATPG set . However, it will be detected if 'true' implementation ATPG set is available. The other undetected Trojan type (EthernetMAC-T100-T600) is a timing Trojan, in which signals timing was modified by manipulation of wire parameters. Although these Trojans are not detectable by a zero-delay logic simulation, in reality, they will most likely affect the at-speed scan behavior. It has to be noted that the Trust-hub benchmarks did not include logic that hides the Trojan logic in scan mode. Moreover, there was no notion of scan mode. We believe that some of the Trojans can be completely hidden from scan given the presence of the scan mode indication.

c) Random netlist deviations: For this part, we used the open-source RTL benchmarks: [33], [34], [35], [36]. For each benchmark, we generated a set of deviated netlists by applying a predefined number of random modifications to the netlists generated from the original benchmark. Each modification is either a random gate removal or a random gate insertion. The original benchmark netlist played the role of the open-source 'golden model', completely available to the validator, and the deviated netlists represented the partial data extracted from Silicon using scan. The subsequent flow was identical to the Trust-hub flow.

Figure 6 shows the success rate of detection of the modifications for different benchmarks using the dependency graph comparison and the ATPG-based equivalence checks versus number of modifications. Each gate insertion or deletion counts as a modification. In all cases, the verification flow successfully identified a deviation in at least one of the stages. Clearly, equivalence check with ATPG vectors has much greater coverage. Together with that, the experiments have shown both steps are essential. Several modifications were captured only by the dependency graph comparison, but not by the ATPG-based logic equivalence check. All such cases had a common property — the deviated netlist only had gates added to the original netlist. The added gates essentially introduce new faults that are not necessarily covered by the ATPG vectors of the original netlist. Recall that we did not include the vectors of the deviated netlist to the verification set, hence the verification may succeed in spite of the difference.

VI. Conclusions and Future Work

The open source hardware IP model promises to improve security, interoperability and reliability of the secure integrated circuits. In this work, we brought to the attention of the community a significant pitfall of the model existing along with its benefits. In contrast to software, conformance of a cryptographic IC to the claimed open source is not self-evident. Therefore, dishonest actors along the

supply chain may make malicious or cost-saving modifications that will get unnoticed by the user. As a solution, we proposed Open Scan, a verification technique that practically adds the missing part of the open-source IC security model. The technique exploits the production scan to perform partial reverse engineering of the device under test and compares the result with the intended source. The IC vendor cooperation is required to obtain scan metadata. The verification comprises several stages, namely a dependency graph generation and comparison, hidden state search and ATPG-based logic equivalence check.

The proposed method does not formally prove equivalence. However, in the experimental part we observed that in practice it was able to detect all the changes randomly inserted in the benchmarks and majority of Trojans from Trust-Hub. We also discussed attacks via Open Scan and a dishonest IC vendor that may provide deceiving data or hide the malicious circuit from scan. The latter is a particularly challenging question that can be addressed by future research. There are additional interesting research questions that can be addressed in the future. One of them is verification of a product, containing both open-source and proprietary logic. We contribute the sources of the Open Scan verification flow to the community [38].

REFERENCES

[1] G. Gupta, T. Nowatzki, V. Gangadhar, and K. Sankaralingam, "Kickstarting Semiconductor Innovation with Open Source Hardware," pp. 50–59, 2017.

[2] "OpenCores," https://opencores.org/.

[3] "OpenPOWER Foundation," https://openpowerfoundation.org/.

[4] "OpenRISC," https://openrisc.io/.

[5] A. S. Leon, K. W. Tam, J. L. Shin, D. Weisner, and F. Schumacher, "A power-efficient high-throughput 32-thread SPARC processor," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 1, pp. 7–16, 1 2007.

[6] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovic, "The risc-v instruction set manual, volume i: Base user-level isa," *EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2011-62*, vol. 116, 2011.

[7] "Open source silicon root of trust (RoT) — OpenTitan," https://opentitan.org/.

[8] "libreSOC," https://libre-soc.org/.

[9] "OpenHW Group," https://www.openhwgroup.org/.

[10] "EPI - European Processor Initiative," https://www.european-processor-initiative.eu/project/epi/.

[11] M. Billmann, S. Werner, R. Holler, F. Praus, A. Puhm, and N. Kero, "Open-source crypto IP cores for FPGAS-overview and evaluation," in *Proceedings - 2019 Austrochip Workshop on Microelectronics, Austrochip 2019*. Institute of Electrical and Electronics Engineers Inc., 10 2019, pp. 47–54.

[12] J. Goldman, "How Secure Are RISC-V Chips?" *Semiconductor Engineering*, 2023.

[13] J. Baehr, G. Sigl, A. Bernardini, and U. Schlichtmann, "Machine learning and structural characteristics for reverse engineering," in *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*. New York, New York, USA: Institute of Electrical and Electronics Engineers Inc., 1 2019, pp. 96–103.

[14] A. Huang, S. Cross, and T. Marble, "media.ccc.de - Open Source is Insufficient to Solve Trust Problems in Hardware," 2019.

[15] H. Li, Q. Liu, and J. Zhang, "A survey of hardware Trojan threat and defense," *Integration, the VLSI Journal*, vol. 55, pp. 426–437, 9 2016.

[16] C. Bao, D. Forte, and A. Srivastava, "On Reverse Engineering-Based Hardware Trojan Detection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, 2016.

[17] A. Miczo, *Digital Logic Testing and Simulation*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 8 2003.

[18] L. Azriel, R. Ginosar, and A. Mendelson, "Revealing On-chip Proprietary Security Functions with Scan Side Channel Based Reverse Engineering," in *Proceedings of the 27th Edition o f the Great Lakes Symposium on VLSI*, vol. Part F1277, 2017.

[19] L. Azriel, R. Ginosar, S. Gueron, and A. Mendelson, "Using Scan Side Channel to Detect IP Theft," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 12, pp. 3268–3280, 12 2017.

[20] D. Brand, "Verification of large synthesized designs," in *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*. IEEE, 1993, pp. 534–537.

[21] S.-Y. Huang and K.-T. Cheng, *Formal Equivalence Checking and Design Debugging*, ser. Frontiers in Electronic Testing. Boston, MA: Springer US, 1998, vol. 12.

[22] V. D. Agrawal, "Choice of tests for logic verification and equivalence checking and the use of fault simulation," in *Proceedings of the IEEE International Conference on VLSI Design*. IEEE, 2000, pp. 306–311.

[23] L. T. Wang *et al.*, *VLSI Test Principles and Architectures: Design for Testability*, ser. Systems on Silicon. Elsevier Science, 2006.

[24] I. Wegener, *The Complexity of Boolean Functions*. John Wiley & Sons, Inc., 1987.

[25] M. Fyrbiak *et al.*, "HAL- The Missing Piece of the Puzzle for Hardware Reverse Engineering, Trojan Detection and Insertion," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2018.

[26] L. Azriel, R. Ginosar, and A. Mendelson, "Exploiting the Scan Side Channel for Reverse Engineering of a VLSI Device," Technion, Israel Institute of Technology, Tech. Rep. CCIT Report 897, 2016.

[27] Y.-T. Lai, C. Chang, K.-C. Chen, and C.-C. Lin, "Combinational equivalence checking methods and systems with internal don't cares," 2007.

[28] J. Da Rolt, A. Das, G. Di Natale, M.-L. Flottes, B. Rouzeyre, and I. Verbauwhede, "A scan-based attack on Elliptic Curve Cryptosystems in presence of industrial Design-for-Testability structures," in *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, 10 2012, pp. 43–48.

[29] D. Hely, K. Rosenfeld, and R. Karri, "Security challenges during VLSI test," in *IEEE 9th International New Circuits and systems conference*. Ieee, 6 2011, pp. 486–489.

[30] 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, 3 2014.

[31] D. Hely, F. Bancel, M. Flottes, and B. Rouzeyre, "Test Control for Secure Scan Designs," *European Test Symposium (ETS'05)*, pp. 190–195, 2005.

[32] B. Shakya *et al.*, "Benchmarking of Hardware Trojans and Maliciously Affected Circuits," *Journal of Hardware and Systems Security 2017 1:1*, vol. 1, no. 1, pp. 85–102, 4 2017.

[33] "Toy RISC-V model for teaching," https://anonymous.4open.science/r/rv-2023/README.md, 2021.

[34] P. D. Schiavone *et al.*, "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for internet-of-things applications," *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation, PATMOS 2017*, vol. 2017-Janua, pp. 1–8, 11 2017.

[35] H. Hsing, "tiny_aes IP project," http://opencores.org/project,tiny_aes, 2012.

[36] Y. Peng, "Bitcoin Double SHA256 project," http://opencores.org/project,btc_dsha256.

[37] M. Fyrbiak *et al.*, "Hardware reverse engineering: Overview and open challenges," in *2017 IEEE 2nd International Verification and Security Workshop (IVSW)*. IEEE, 7 2017, pp. 88–94.

[38] "OpenScan Repository - Anonymous GitHub," https://anonymous.4open.science/r/Openscan-7346/README.md, 2023.