On Hardware Trojan Detection using Oracle-Guided Circuit Learning

Rajesh Kumar Datta, Guangwei Zhao, Dipali Deepak Jain, Kaveh Shamsi
{rajesh.datta,guangwei.zhao,dipali.jain,kaveh.shamsi}@utdallas.edu
ECE Department, University of Texas at Dallas
Richardson, Texas, USA

ABSTRACT

Hardware Trojans, i.e. malicious circuitry inserted into a design by an untrusted foundry or designer, pose a threat to the fabless semiconductor industry. The detection of hardware Trojans has been the subject of numerous studies over the years. In this paper, we discuss a novel approach to Trojan detection: using the framework of oracle-guided circuit learning (OGCL) or deobfuscation, which has traditionally been used for assessing the security of circuit obfuscation schemes. We show how arbitrary functional Trojan detection can polynomially be reduced to OGCL, yielding a more formal and versatile framework than traditional heuristic techniques. This formulation can also be used to locate Trojans and can be easily extended to side-channel or hybrid detection by using non-functional OGCL. The main challenge with this approach is its worst-case-exponential space complexity when using baseline Boolean satisfiability (SAT)-based circuit deobfuscation. To this end, we propose some novel techniques based on AllSAT, cube generalization, and quantified Boolean Formula (QBF) solving. We present a set of experiments on benchmark circuits to showcase the validity and performance of our framework.

CCS CONCEPTS

• **Security and privacy → Malicious design modifications**.

KEYWORDS

Hardware Trojans, Oracle-guided Circuit learning

ACM Reference Format:
Rajesh Kumar Datta, Guangwei Zhao, Dipali Deepak Jain, Kaveh Shamsi. 2024. On Hardware Trojan Detection using Oracle-Guided Circuit Learning . In *Great Lakes Symposium on VLSI 2024 (GLSVLSI '24), June 12–14, 2024, Clearwater, FL, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649476.3658705

1 INTRODUCTION

The globalization of Integrated Circuit (IC) production has introduced some new security and privacy concerns. The disclosure of the physical design to untrusted parties, hardware Trojans are the major two. Hardware Trojans are malicious modifications of the design at some point in the production process with the aim of altering the original functionality, reducing performance, or leaking critical information [13]. Researchers have been addressing the issue of hardware Trojans using various techniques over the years. These techniques mainly aim to either detect the Trojan or make it harder to insert Trojans into the design in the first place.

One of the primary techniques for post-silicon non-invasive (i.e. not involving physical reverse engineering) Trojan detection is functional testing. Stealthy Trojans are generally designed to activate under rare specific conditions, and as such attach to rare nets/nodes of the circuit. Logic testing for Trojan detection aims to activate these rare nets by testing vectors not typically visited by traditional automatic test pattern generations (ATPG). Applying such patterns to the primary inputs and observing the output to detect any mismatches can help identify Trojan-infested chips[3]. Various specialized ATPG, model checking-based [5] and SAT-based approaches have been used to this end. Power and electromagnetic (EM) side channels [7] and laser probing have also been used as semi-invasive post-silicon techniques.

In this work, we have explored a novel approach to Trojan detection: using the framework of oracle-guided circuit learning (OGCL) (or deobfuscation) to detect Trojans. Oracle-guided circuit learning is the problem of finding a set of unknowns/keys in an ambiguous circuit by making adaptive queries to an oracle of that circuit. This problem has been studied in the circuit obfuscation domain for some years now, in which an attacker tries to recover the full design of an ambiguous obfuscated circuit by modeling the ambiguity as unknowns in a circuit learning problem. SAT-based techniques here have been shown to be very promising [6, 12]. Along this line, the paper delivers the following novel contributions:

- We formulate golden-IC-free Trojan detection as an oracle-guided circuit learning problem. We show how using probable Trojan models (PTM): circuit gadgets that can capture the potential existence of many Trojans in the circuit, a circuit learning procedure can be used to prove or disprove not just the existence of such a Trojan, but also their potential location and activation conditions.
- The main challenge with off-the-shelf circuit learning procedures for use in Trojan detection is their linearly growing space complexity (in the number of queries) when applied in this way. We propose several novel techniques including using AllSAT solving, cube generalization and merging, and Quantified Boolean Formula (QBF) solving.
- We implement our algorithms and evaluate them against a set of benchmark circuits and present the results showing an alignment with our theoretical analysis.

The rest of the paper is organized as follows: Section 2 provides preliminaries, Section 3 provides the main methodology, Section

GLSVLSI '24, June 12–14, 2024, Clearwater, FL, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0605-9/24/06
https://doi.org/10.1145/3649476.3658705

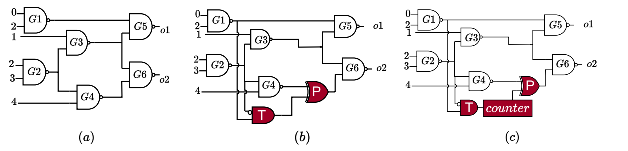


Figure 1: Trojan example: (a) Trojan-less circuit (b): combinational Trojan with AND/fixed-comparator trigger and an XOR-based payload. (c) Sequential Trojan n-bit counter-based trigger with XOR Payload.

4 presents methods for space efficiency, Section 5 presents experiments and Section 6 concludes the paper.

2 PRELIMINARIES

Trojan Model: The original/golden circuit $c_g(x) : X \longrightarrow Y$ implements a Boolean function from the n-bit input space X to the m-bit outputs space Y, and has a circuit graph G_g. A gate-level hardware Trojan is a malicious modification of the circuit graph G_g to G_t. This typically involves a trigger function $T(w)$ attached to some nets w, and a payload P. If the payload alters the functionality of c_g at its observable functional outputs to c_t, then the Trojan is *active*. Fig. 1 shows examples of combinational and sequential Trojans. More details on hardware classification can be found in [13].

Signal Probability Estimation and Rare Nets. To have a stealthy Trojan it needs to activate in rare specific conditions. To achieve this the attacker typically needs to attach the trigger to signals with low transition probabilities [8]. For modeling potential Trojans, this fact needs to be kept in mind. Calculating the precise signal probabilities for the internal nets is #P-complete (practically even harder than NP-complete problems). One can approximate signal probabilities with pattern simulation, or propagating probability values under the (wrong) assumption of statistical independence for inputs of each gate in the circuit and propagation rules [9].

Oracle-Guided Circuit Learning (OGCL). Given an obfuscated/ambiguous circuit $c_e : K \times X \longrightarrow Y$ with l hidden/"key" inputs $K = \{0, 1\}^l$ and n primary inputs $X = \{0, 1\}^n$ and m outputs $Y = \{0, 1\}^m$, and an oracle/target circuit $c_o : X \longrightarrow Y$, oracle-guided circuit learning/deobfuscation is the problem of finding a value for the hidden inputs k_* such that the ambiguous circuit becomes equivalent to the oracle $\forall x \in X \;\; c_e(k_*, x) = c_o(x)$. In the sequential version of this problem, c_e and c_o can be sequential circuits. This problem has many applications. Specifically, in hardware security it is used extensively to study the security of logic obfuscation schemes [6, 11, 12].

SAT-based OGCL. A powerful framework for generic combinational circuit learning is the SAT-based approach first presented in [6, 12]. Here the attacker/learner proceeds by building a miter condition $M = (c_e(k_1, x) \neq c_e(k_2, x))$. This condition is converted to a conjunctive-normal-form (CNF) SAT problem and solved using a modern SAT solver. If M is satisfiable using some $\hat{x}, \hat{k_1}, \hat{k_2}$, this means that an input pattern $\hat{x}$ exists for which there are two possible keys that produce different outputs, i.e. there is ambiguity in the learner's hypothesis on $\hat{x}$. This input is queried on the oracle, $\hat{y} = c_o(\hat{x})$. The resulting input-output observation is appended to M as a constraint. $F \longleftarrow M \wedge [(c_e(k_1, \hat{x}) = \hat{y}) \wedge (c_e(k_2, \hat{x}) = \hat{y}))]$. The process continues until the constraints+miter condition F becomes UNSAT. At this point, there is no ambiguity left over the functionality of c_e and a provably functionally correct key can be extracted from F. Algorithm 1 shows the procedure.

Algorithm 1: Given an oracle c_o and the locked net list c_e returns a correct key $k_* \in K_*$ if $c_o \in \{c_e(k, .)|k \in K\}$.

```
Function SATOGCirLearn($c_e$, $c_o$ as black-box oracle):
    $F \leftarrow true$
    $M \leftarrow c_e(k_1, x) \neq c_e(k_2, x)$
    while $F \wedge M$ is satisfiable do
        $\hat{x}, \hat{k_1}, \hat{k_2} \leftarrow$ SAT($F \wedge M$)
        $\hat{y} \leftarrow c_o(\hat{x})$
        $F \leftarrow F \wedge (c_e(k_1, \hat{x}) = \hat{y}) \wedge (c_e(k_2, \hat{x}) = \hat{y})$
    $\hat{k_1}, \hat{k_2} \leftarrow$ SAT($F$)
    return $\hat{k}_1$ as correct key $k_*$
```

Power OGCL. In [10] PowerSAT was proposed which is an OGCL algorithm for learning circuits from power side-channel (hamming sum) queries rather than functional ones. The algorithm operates similarly to the functional case except the functional miter is replaced with a power model analog which returns power-discriminating input patterns (PDIPs). These create disagreements between the (hamming sum) power model (which are psuedo-Boolean expressions) for different keys. Input-power pairs are reinforced back into the solver. A differential version, PowerSATdiff, was also proposed which mines for pairs of power queries via a differential miter and uses this observed power comparison direction as constraints on the key. This improves the robustness of the procedure to hamming sum querying noise.

3 TROJAN DETECTION VIA OGCL

The post-silicon functional Trojan detection task can be mapped to oracle-guided circuit learning (OGCL) as follows: The defender has a chip c_t that he suspects may have a Trojan inside. He also has the original golden design c_g (not a golden IC). The defender can detect a potential Trojan, by first building a circuit model $c_e(k, x)$, for which *the choice of the unknown/key inputs k captures the presence or absence of a particular Trojan in the circuit.* It follows that there must exist a hidden/key variable assignment k_* for which $\forall x \;\; c_e(k_*, x) = c_g(x)$, i.e the model must include the golden circuit. This is so that OGCL run against a Trojan-free circuit $c'_t \equiv c_g$ will properly conclude returning a key indicating the absence of Trojans. If the OGCL procedure is instead run against a Trojan-infested design c_t as the oracle instead of c_g, the hope is that this will lead to some discernible deviation in the process.

3.1 Possible Trojan Model (PTM)

The keyed circuit model $c_e(k, x)$ here is built in a way such that the choice of the key k captures the presence or absence of a Trojan. How to construct such a circuit? Since there is not just one possible Trojan to capture/model, there are many ways to construct many different such $c_e(k, x)$s. We propose to do this by taking the original design c_g, and inserting circuit gadgets that we term possible/probable Trojan models (PTMs) into it. An example of a PTM is seen in Fig. 2a. Here by setting the key variable k_{act} to 0, the original circuit's functionality is unchanged. If k_{act}=1, then on a particular combination of the input nets R, the comparator will trigger and as a result, a wire value will flip. So the choice of *kact*

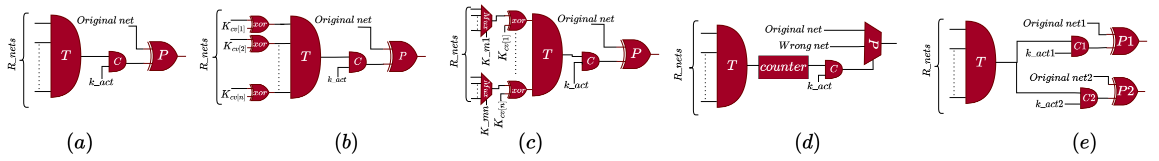


Figure 2: Possible Trojan models (PTMs). (a) captures a fixed comparator potentially unleashing a payload on the original net when k_{act} = 1 (b) comparator to an unknown value, k_{act} = 1 indicates presence, and the K_{cv} vectors indicate the activation location. (c) unknown-value comparator with an unknown location. K_{mn} indicate input location. (d) combination of a comparator with sequential counter and MUX payload. (e)Multi-payload PTM composition.

models whether a comparator-triggered Trojan is present on those particular wires or not. We can develop an array of such PTMs:

Fixed Comparator: Seen in Fig. 2a models the presence of a Trojan that checks to see if a set of input wires R_{nets} match a fixed pattern, and if so triggers a single XOR payload. This takes up a single hidden/key variable k_{act} which if equal to 1, indicates the presence of such a Trojan. The query complexity for detecting a single such Trojan is $O(1)$.

Unknown-Value Comparator: Seen in Fig. 2b, models not just the presence of a comparator Trojan, but also, the value that is being compared to is left as an unknown vector k_{cv}. Therefore, any comparator (comparison with any vector k_{cv}) Trojan that is present in the oracle, that is connected to the same input wires as this PTM can be modeled and therefore detected using OGCL. The query complexity of provably detecting such a Trojan is at most $O(2^n)$.

Unknown-Value and Unknown-Span Comparator: Seen in Fig. 2c, models a comparator with an unknown value connected to an unknown subset of the wires R_{nets} of size n. The connection ambiguity is modeled by key-controlled MUX gates added to the inputs of the comparator.

Sequential Counter: Seen in Fig. 2d shows an example of a sequential counter PTM. This captures the presence of a Trojan that can count a certain number of activations on a particular comparator and trigger the payload. The inclusion of such a PTM in the circuit makes the c_e model sequential, therefore necessitating a sequential OGCL procedure such as the model-checking attack in [11].

Multi-Payload PTM As shown in Fig. 2e, a PTM can have multiple payload locations that activate based on key bits. This can help model the ambiguity in (and determine via OGCL) the location of the possible payload in the Trojan-infested circuit. One can design a large PTM with many potential payload positions. p different payload locations can often increase the query complexity of the detection by p times, although this is not a rule.

PTM Span. As seen in the above examples, any PTM has a known possible function space. Each PTM can model a certain set of Trojans present in the circuit. Since the exact Trojan is unknown to the learner, the larger and more expressive the space of possible Trojans can be for c_e, the higher the success probability of the OGCL Trojan detection procedure (at the cost of higher query complexity).

3.2 SAT-based Circuit Learning against PTMs

Trojan within PTM Span. We now explore what happens when an OGCL procedure such as the SAT-attack from Algorithm 1 is run against a c_e, c_t pair of circuits. First, consider the case where the Trojan happens to fall in the space of possible functions of c_e. i.e. $c_t(.) \in \{c_e(k,.)|k \in K\} = C_e$. Here, the SAT-based OGCL goes through its iterations, interacting with the c_t chip in an adaptive manner and lands on a candidate key that makes the miter UNSAT.

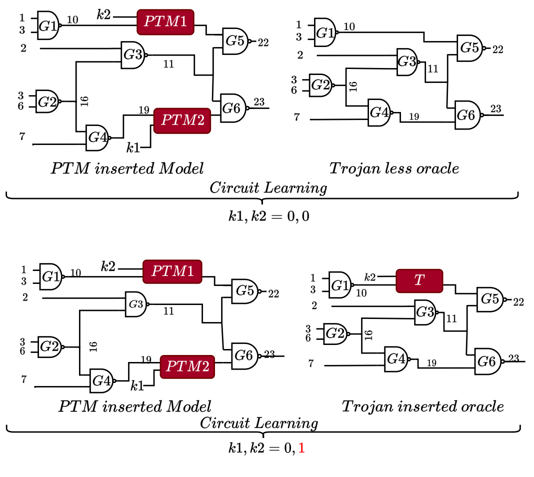


Figure 3: The returned keys from the OGCL procedure will indicate whether a Trojan was present in the circuit in a particular location (bottom) or not (top).

The io-constraints will be satisfiable and a provably functionally correct key will be extracted. We can take this key and analyze it. For instance, if a k_{act} bit in the key corresponds to the activation of a particular PTM, and that bit happens to be 1 when returned by the OGCL, this could indicate a Trojan presence.

Does this prove the existence of a Trojan? The answer is yes except for one case. The only case in which an attack may return a k_{act}=1 on a Trojan-free circuit, is if the PTM happens to not impact the observable (output in functional, and power in power OGCL respectively). This can occur if the PTM's output is triggered on some condition, but this output does not propagate to an observable.

The above case, however, can be avoided in two ways: 1) we ensure to insert PTM payloads that do in fact alter the circuit's functionality. 2) we can perform a looseness check to see if both 0 and 1 are possible choices for a given activation key bit without violating the io-constraints F.

We can generalize the above into the following lemma:

Lemma 3.1. *If the circuit under test is Trojan-infested and has functionality c_t where $c_t \in \{c_e(k,.)|k \in K\}$ where c_t deviates from the original functionality c_g, and $c_g \in \{c_e(k,.)|k \in K\}$, then the SAT-based OGCL procedure will terminate properly with a correct key that can help determine if the circuit under test is functionally equivalent to the golden design c_g.*

Trojan outside PTM Span. Now let us consider the converse case. When the Trojan-infested chip has functionality c_t not modeled by the PTM $c_t \notin C_e$. Here, there are two possible outcomes:

1) The SAT-based OGCL concludes with an UNSAT miter+io-constraints and terminates. However, when the correct key is to be

Algorithm 2: Given an circuit under test c_t and the golden circuit model c_g tries to detect likely Trojans.

```
Function OGCLTrojDetect($c_g$ golden model, $c_t$ as black-box):
    for $i \in$ Rounds do
        $c_e \leftarrow$ InsertLikelyPTMs($c_g$)
        $F \leftarrow true$
        $M \leftarrow c_e(k_1, x) \neq c_e(k_2, x)$
        while $F \wedge M$ is satisfiable do
            $\hat{x}, \hat{k_1}, \hat{k_2} \leftarrow$ SAT($F \wedge M$)
            $\hat{y} \leftarrow c_t(\hat{x})$
            $F \leftarrow F \wedge (c_e(k_1, \hat{x}) = \hat{y}) \wedge (c_e(k_2, \hat{x}) = \hat{y})$
        if $F$ satisfiable with $k_*$ then
            $TrojPresent, TrojLoc \leftarrow$ AnalyzeKey($k_*$)
        else
            UNSAT constraints.
            $TrojPresent \leftarrow true$
```

extracted in the final step from the io-constraints F, the solver returns an UNSAT status. i.e. there exists no key k that can satisfy the io-constraints $\bigwedge_{i \in q}(c_t(x_i) = c_e(k, x_i))$. In such a case, if the PTM model was constructed properly, this is proof of Trojan presence. The reason is simple. The PTM model c_e was constructed in a way as to include the original functionality, i.e. $c_g \in C_e$. Therefore, a SAT-based OGCL interacting with $c_t \equiv c_g$ would never result in an UNSAT io-constraint. This must mean that $c_t \neq c_g$. i.e. the OGCL procedure is not talking to a function equivalent to the golden circuit c_g.

2) The trickier case with a Trojan outside of C_e is when the SAT-based OCGL concludes early with an UNSAT miter, and a SAT io-constraint and a key. This can happen due to the fact that there can exist input-output constraints $c_t(x_i) = c_e(k, x_i)$, where even though $c_t \notin C_e$, the constraints are still satisfiable, it is just that the constraints knock the miter UNSAT early. In such a case the returned key can be studied. If it suggests a Trojan presence then a Trojan presence is proven. If however, the returned key does not suggest a Trojan presence, this is not proof of Trojan absence. In this case, other than noticing the early termination by its large deviation from the expected query count for a PTM class, we can have a true false negative that cannot be avoided without using more expressive PTMs.

The overall flow for the procedure can be seen in Algorithm 2.

3.3 Intelligent PTM Insertion

We now know that the more expressive and the larger the space of C_e the lower the probability of Trojans evading the detection. At the same time, the more expressive the PTM, the more work the SAT-based OGCL will have to perform. Note that certain PTMs can raise the query complexity of SAT-based OGCL exponentially. Unknown-value comparator PTMs with n inputs, yield OGCL query-complexities of $O(2^n)$ (this is similar to the deobfuscation of point-function schemes[15]). Note however, that this is not a unique issue of our approach, but that simply the minimum query complexity of detecting combinational n-bit comparator Trojans with unknown activation patterns is $O(2^n)$ using any functional testing method.

As such, in the detection process, it is important to insert PTMs prudently. In our proof-of-concept experimentation, we use signal probability estimation to find rare nets. We attach PTMs to these rare nets to capture more plausible Trojans.

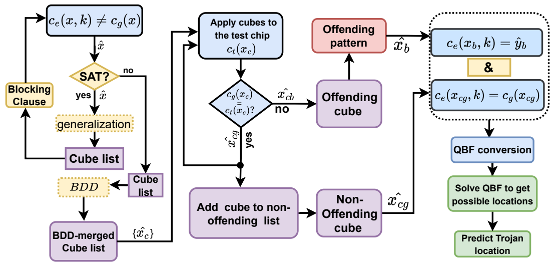


Figure 4: Flow of Semi online OGCL: From trojan pattern and cube generation to trojan detection test and localize the trojan with QBF using PTM.

3.4 Multiple PTM Composition

One can insert multiple PTMs into the circuit to capture multiple types of Trojans in the same circuit model c_e. Note that each PTM has key bits that capture its absence. That means that inserting a new PTM does not violate the $c_g \in C_e$ inclusion rule, therefore, as long as a Trojan-infested chip's behavior c_t falls within C_e, detection will be guaranteed in finite time.

When multiple PTMs are used, the OGCL procedure will return a longer key vector whose various bits belong to different PTMs. It is possible to study these bits individually and try to use this information to determine where the Trojan was inserted. Per Fig. 3, two PTMs are inserted into the circuit with activation key bits k_1 and k_2. If the OGCL returns k_2=1 after interacting with the circuit under test, this can mean that the corresponding PTM's Trojan was present but not the other. For the outside-span case $c_t \notin C_e$, such determinations cannot be made conclusively. It is possible for outside-span Trojan behavior to end up asserting the activation key bits of a differently located PTM.

4 IMPROVING SPACE EFFICIENCY WITH SEMI-ONLINE OGCL

The main challenge with using the SAT-based OGCL in Algorithm 1 in our flow is its space complexity. The baseline SAT-based OGCL algorithm repeatedly appends circuit copies to the CNF causing it to grow linearly in size. For n bit PTMs this leads to an exponential $O(2^n)$ space complexity, which is a lot more problematic than exponential time complexity. Note that the more advanced KC2 OGCL algorithm [11] which simplifies key conditions on the fly can help here. But we propose a couple of additional solutions herein.

One reason for the growth in the space usage of the SAT-based OGCL is its online nature. Each query is immediately appended to the solver and a new query is generated via another SAT call. We can break this slightly, and instead generate a ton of patterns with a given CNF snapshot before extending it with additional constraints.

AllSAT Pattern Enumeration. We can create a new miter (see Fig. 4) one that compares the output of the PTM-inserted circuit with the original golden circuit $c_e(k, x) \neq c_g(x)$. In the extreme, we can use an all-solutions SAT (AllSAT) [14] loop to generate all the patterns x for which c_e can deviate from c_g. Testing the circuit on all these patterns and observing no deviation proves in-span Trojan absence. This immediately cools space complexity, as modern AllSAT solvers have sub-exponential space complexities.

The loss of the fully online procedure is not a great disadvantage, as both techniques detect within span Trojans.

Pattern to Cube via Generalization. In addition to AllSAT, we can use cubes instead of individual patterns. Cubes here are input vectors with "don't-care" bits. e.g. 00xx, which is a single cube that captures 4 different input patterns. Once we satisfy the above miter with $\hat{x}, c_e(k, \hat{x}) \neq c_g(\hat{x})$, we can use the method of *cube generalization* used extensively in model-checking [4] to go from this concrete pattern $\hat{x}$ to a cube $\hat{x_c}$ that satisfies the same constraint. Generalization techniques typically work by inverting the condition making it UNSAT, and then trying to drop literals from a set of solver assumptions to see if the UNSAT status of the problem remains. UNSAT-Cores returned by modern SAT solvers, which capture literals that were relevant to the UNSAT result are used here. We use a procedure similar to the one in [4] in our experiments. Once a cube (e.g. $x = 00d$) is generated we can ban it from the solver by adding a ban-clause $(\overline{x_1} + \overline{x_2})$ and repeating the process. This procedure is in fact a form of accelerated AllSAT enumeration.

Cube Merging. After a cube is generated, it captures a set of test vectors. We can merge multiple cubes into bigger (looser, with more don't care bits) cubes using cube minimization algorithms. e.g. : $00d$ and $01d$ can be merged into $0dd$. We use a Binary Decision Diagram (BDD) engine for this where cubes are added to a BDD which automatically merges them, and then all the BDD minterms are enumerated which returns the merged cube list. The final set of cubes can represent an exponential number of patterns that can be individually queried in a batch fashion.

Trojan Localization using Cube Conditions and QBF. During testing, if a deviation from the original circuit is detected we can use this offending input-output constraint to localize the Trojan. However, including non-offending cubes which agree with the golden design, into the constraint can improve the accuracy of this localization. We can include a cube constraint of the form e.g. $c_e(k, 00dd) = 100$ if we use a Quantified Boolean Formula (QBF) solver instead of the baseline SAT solver. QBF solvers allow for specifying universal quantifiers, hence the above constraint becomes $\forall x_1, x_2\ c_e(k, 00x_1x_2) = 100$. QBF solving is much more complex than SAT but modern QBF solvers perform quite well here as we will see in our experiments. Figure 4 represents the flow of this semi-online method from pattern generation to Trojan detection and localization.

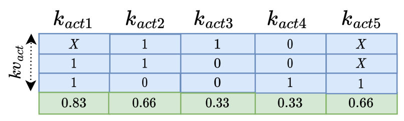
Localization by Activation Key Cube Analysis. Our QBF solver can return an enumerated list of cube solutions to the key k from the above observation condition. Per Fig. 5 we can study the activation key bits in these key cubes to derive the probability of Trojan presence, by counting the number of 1s and 0s under each activation key bit over the set of cubes while keeping track of don't-care bits. A probability of 1 or 0 proves presence or absence at a particular location. Intermediate values not equal to 0.5 capture non-zero information learned about the Trojan location.

5 EXPERIMENTS

We implemented our proposed methods in C++ and Python on a set of different benchmark circuits. We detected rare nets using signal probability rule propagation. We used `neos` [2] for SAT and PowerSAT OGCL. Tests were run on single threads of an Intel i7 3GHz 16GB memory machine running Ubuntu 20.

Online OGCL: We first evaluated the online SAT-based OGCL against Trojan-free circuits. This for n-bit p-payload comparators yielded a routine $O(p.2^n)$ (point-function deobfuscation) query complexity which is in line with theory. Of course, false positive rates for in-span Trojans here were zero. We then ran the OGCL procedure with the Trojan-infested c_t and multi-payload PTMs. In this case, we observed a 100% Trojan presence detection upon termination. Trojan location detection as measured by activation key bits that were set to 1 was not 100%. This could be due to different Trojans masking each other's locations, or some Trojans not having an impact on the output functionality in our proof-of-concept implementation. In Table 1 SAT-OGCL Trojan detection time and required iteration for different Trojan-inserted benchmarks against different PTM models are reported.

We also experimented with power side-channel-based OGCL for Trojan detection using PowerSATeq and PowerSATDiff from the framework of [2]. We generated fixed comparators of different sizes and patterns as the c_g. Then c_e was instantiated as an unknown-value comparator. Following the same observations from [10] PowerSAT attack recovered n-bit vectors from comparators. These values were then compared to the golden model's comparison vector with a 100% detection accuracy. In Table, 2 Trojan detection with PowerSAT results have been reported.



kv_{act}	k_{act1}	k_{act2}	k_{act3}	k_{act4}	k_{act5}
	X	1	1	0	X
	1	1	0	0	X
	1	0	0	1	1
	0.83	0.66	0.33	0.33	0.66

Figure 5: Probability calculation for a PTM payload (k_{act} having trojan using the results from QBF-based solving.

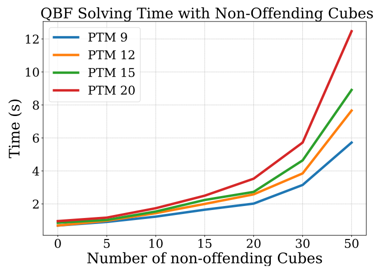


Figure 6: Runtime of the QBF solver for c432 as more non-offending cubes are added and as the number of possible PTM payloads increases.

Semi-Online OGCL: We implemented the semi-online OGCL flow from section 4 and figure 4. After extracting each pattern, it

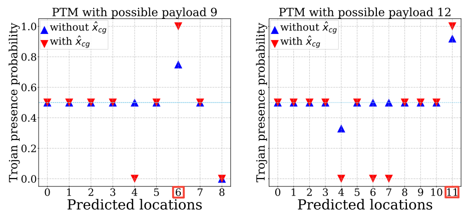


Figure 7: Measurement of Trojan presence probability from the enumerated QBF solutions. The red box on the x-axis indicates the true Trojan location, whose probability went up after including good cubes.

			True Payload location 1								True Payload location 2								True Payload location 3							
			3inp-tri		4inp-tri		6inp-tri		8 inp-tri		3inp-tri		4inp-tri		6inp-tri		8 inp-tri		3inp-tri		4inp-tri		6inp-tri		8 inp-tri	
Bench	R_{net}	Possible Payloads	T(s)	iter	T(s)	iter	T(s)	iter	T(s)	iter	T(s)	iter	T(s)	iter	T(s)	iter	T(s)	iter	T(s)	iter	T(s)	iter	T(s)	iter	T(s)	iter
c432	8	10	0.17	55	0.15	45	0.19	55	0.19	65	0.14	42	0.11	30	0.18	55	0.16	55	0.09	14	0.1	16	0.18	55	0.11	24
	9	20	0.11	21	0.13	23	0.17	63	0.20	65	0.13	46	0.19	81	3.92	407	8.88	584	0.12	46	0.14	62	0.15	62	1.22	241
	10	30	0.40	32	10.5	409	15.7	507	21.7	614	3.48	238	7.81	367	44.1	841	121	1467	3.04	229	1.31	153	1.50	153	4.95	303
c880	8	10	0.75	86	0.98	129	4.89	315	8.89	415	1.95	219	9.26	431	552	1378	4.51	345	0.27	32	0.29	29	8.82	428	5.46	346
	9	20	0.86	115	1.19	115	1.27	125	1.23	125	0.14	9	0.20	13	183	1622	222	1885	0.19	11	0.24	15	0.46	52	0.71	87
	10	30	18.1	429	18.1	429	313	1825	314	1825	0.18	23	16.5	47	3.88	25	128	1355	0.30	20	9.32	360	0.76	79	14.2	455
c1908	8	10	0.39	9	0.36	15	10.6	184	8.64	184	0.34	7	0.52	6	9.80	184	10.5	184	0.52	5	0.30	14	10.7	184	8.14	184
	9	20	0.40	7	1.14	80	43.4	203	37.3	243	0.82	60	1.20	83	40.3	243	41.5	243	0.37	18	2.52	125	40.6	243	36.6	243
	10	30	0.48	16	0.42	7	206	347	198	347	0.45	23	0.50	10	214	347	216	347	0.30	4	0.44	8	200	327	190	347
c2670	8	10	6.89	354	2.60	216	153	1519	162	1619	24.8	672	21.8	632	50.8	864	164	1619	3.11	237	44	961	1.70	165	152	1619
	9	20	3.26	262	21.9	689	15.4	599	317	2635	3.13	279	8.68	429	0.72	76	1220	to	4.73	276	16.8	598	35.2	750	35	750
	10	30	109	899	130	1037	574	2325	1284	3451	85	863	11.4	312	123	982	95	905	109	899	130	1037	574	2325	1284	3451
c5315	8	10	1.16	6	1.47	38	6.69	261	2.13	41	1.06	8	1.68	42	8.34	359	2.80	119	1.02	6	1.41	42	7.86	359	2.77	119
	9	20	1.45	16	1.48	16	2.26	56	2.14	52	1.09	25	2.07	85	10.9	353	3.80	171	1.37	16	1.27	21	2.34	85	3.91	171
	10	30	2.74	87	2.36	87	95	871	62	681	1.01	15	1.63	26	9.15	266	15.2	383	1.18	15	1.36	26	8.61	265	16.1	383

Table 1: Results of online SAT-based OGCL for detecting Trojans in ISCAS combinational benchmark circuits. R_{nets} is the number of rare nets input to the comparator PTM, each having multiple possible payloads. Three tests were conducted for each benchmark circuit by inserting the Trojan in three different positions (True Payload) with varying trigger sizes for the Trojan. Time (T) and required iteration (query) counts are reported. The runtime limit was set to 30 minutes for each benchmark. 'to' represents time-out cases.

	Comparator size=9		PowerSATdiff		PowerSATeq	
Bench	Gates	in/out	T(s)	iter	T(s)	iter
c432	160	36/7	5.3	52	1.89	59
c880	383	60/26	51	72	16	109
c2670	1193	157/63	120	56	32	96
c1908	880	33/25	110	42	12.94	50
c1355	546	41/32	26	11	5.2	11
c5315	2307	178 /123	to	13	9.22	9
c3540	1669	50/22	190	17	11	20

Table 2: Trojan detection using PowerSAT-OGCL. As can be seen PowerSAT query counts (iterations) are much lower than the $O(2^n)$ for the functional case.

			Functional-miter				Power-miter			
Bench	R_{nets}	Payloads	Patterns	cubes	T(s)	red.(%)	Patterns	cubes	T(s)	red.(%)
c432	9	9	512	107	1.01	79.10	512	51	0.49	90.01
c880	9	20	257	79	0.87	69.26	512	32	0.77	93.75
c3540	14	13	972	284	15.02	70.78	1024	85	9.66	91.69
c2670	10	28	324	217	6.20	33.02	1024	183	30.08	82.12
c1908	15	28	680	400	15.95	41.17	512	93	3.00	81.83
c1355	9	10	512	170	0.34	66.79	512	109	1.49	79.17
c5315	12	26	1024	210	9.14	79.49	4096	105	22.61	97.43

Table 3: Statistics and runtime for testing patterns and cube generation for different benchmarks for functional and power miters. Reduction percentage for pattern to cube compression are shown.

is generalized, and then banned in the solver by adding a clause, until all cubes are enumerated. Then the cubes are merged using cudd's BDD engine. The number of cubes before and after merging is reported in Table 3. We see a very big reduction in patterns when merged into cubes for both functional and power miter-based pattern generation.

We took these cubes for the c432 circuit and tested them against the Trojan infested (c_t) which yielded a perfect detection accuracy for the in-span Trojan. As for localization, Fig. 6 and Fig. 7 show the runtime and accuracy of Trojan localization as non-offending cubes are inserted into the equation using the dynqbf [1]QBF solver which can return enumerated cube solutions which can be analyzed for Trojan localization.

6 CONCLUSION

In this work, we presented a novel approach to Trojan detection and localization using oracle-guided circuit learning from both patterns and power and improved their space efficiency by using AllSAT and QBF techniques. These enable the reuse of OGCL tools/research in the hardware Trojan detection problem, in addition to being more expressive and powerful than simple ATPG-based, heuristic, or pure machine-learning alternatives.

ACKNOWLEDGMENTS

This work was supported by a grant from the National Science Foundation (NSF-2155189).

REFERENCES

[1] [n. d.]. dynqbf. https://github.com/TU-Wien-DBAI/dynqbf.

[2] [n. d.]. neos. http://www.bitbucket.com/kavehshm/neos.

[3] Amin Bazzazi, Mohammad Taghi Manzuri Shalmani, and Ali Mohammad Afshin Hemmatyar. 2017. Hardware Trojan detection based on logical testing. *Journal of Electronic Testing* 33 (2017), 381–395.

[4] Aaron R Bradley. 2011. SAT-based model checking without unrolling. In *International Workshop on Verification, Model Checking, and Abstract Interpretation.* Springer, 70–87.

[5] Jonathan Cruz, Farimah Farahmandi, Alif Ahmed, and Prabhat Mishra. 2018. Hardware Trojan detection using ATPG and model checking. In *2018 31st international conference on VLSI design and 2018 17th international conference on embedded systems (VLSID).* IEEE, 91–96.

[6] Mohamed El Massad, Siddharth Garg, and Mahesh V Tripunitara. 2015. Integrated Circuit (IC) Decamouflaging: Reverse Engineering Camouflaged ICs within Minutes.. In *Network and Distributed System Security Symposium (NDSS).*

[7] Fakir Sharif Hossain, Michihiro Shintani, Michiko Inoue, and Alex Orailoglu. 2018. Variation-aware hardware Trojan detection through power side-channel. In *2018 IEEE International Test Conference (ITC).* IEEE, 1–10.

[8] Anindan Mondal, Shubrojyoti Karmakar, Mahabub Hasan Mahalat, Suchismita Roy, Bibhash Sen, and Anupam Chattopadhyay. 2022. Hardware Trojan Detection using Transition Probability with Minimal Test Vectors. *ACM Transactions on Embedded Computing Systems* 22, 1 (2022), 1–21.

[9] Hassan Salmani, Mohammad Tehranipoor, and Jim Plusquellic. 2011. A novel technique for improving hardware trojan detection and reducing trojan activation time. *IEEE transactions on very large scale integration (VLSI) systems* 20, 1 (2011), 112–125.

[10] Kaveh Shamsi and Yier Jin. 2021. Circuit Deobfuscation from Power Side-Channels using Pseudo-Boolean SAT. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD).* IEEE, 1–9.

[11] Kaveh Shamsi, Meng Li, David Z Pan, and Yier Jin. 2019. KC2: Key-Condition Crunching for Fast Sequential Circuit Deobfuscation. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, 534–539.

[12] Pramod Subramanyan, Sayak Ray, and Sharad Malik. 2015. Evaluating the security of logic encryption algorithms. In *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust.* IEEE, 137–143.

[13] Mohammad Tehranipoor and Farinaz Koushanfar. 2010. A survey of hardware trojan taxonomy and detection. *IEEE design & test of computers* 27, 1 (2010), 10–25.

[14] Takahisa Toda and Takehide Soh. 2016. Implementing efficient all solutions SAT solvers. *Journal of Experimental Algorithmics (JEA)* 21 (2016), 1–12.

[15] Muhammad Yasin, Abhrajit Sengupta, M Ashraf, M Nabeel, J Rajendran, and Ozgur Sinanoglu. 2017. Provably-Secure Logic Locking: From Theory To Practice. In *Proc. ACM Conf. on Computer & Communications Security.*