Scalable Activation of Rare Triggers in Hardware Trojans by Repeated Maximal Clique Sampling

Yangdi Lyu and Prabhat Mishra, *Senior Member, IEEE*

***Abstract*—Hardware Trojans are serious threat to security and reliability of computing systems. It is hard to detect these malicious implants using traditional validation methods since an adversary is likely to hide them under rare trigger conditions. While existing statistical test generation methods are promising for Trojan detection, they are not suitable for activating extremely rare trigger conditions in stealthy Trojans. To address the fundamental challenge of activating rare triggers, we propose a new test generation paradigm for Trigger Activation by Repeated Maximal Clique sampling (TARMAC). The basic idea is to utilize a satisfiability modulo theories (SMT) solver to construct a test corresponding to each maximal clique. This paper makes three fundamental contributions: (1) it proves that the trigger activation problem can be mapped to clique cover problem, and the test vectors generated by covering maximal cliques are complete and compact, (2) it proposes efficient test generation algorithms to activate trigger conditions by repeated maximal clique sampling, and (3) it outlines an efficient mechanism to run the clique sampling in parallel to significantly improve the scalability of our test generation framework. Experimental results demonstrate that our proposed approach is scalable and it outperforms state-of-the-art approaches by several orders-of-magnitude in detecting stealthy Trojans.**

***Index Terms*—Trigger activation, clique cover, random sampling, test generation, Trojan detection.**

I. Introduction

Due to increasing complexity and stringent time-to-market constraints, SoC supply chain involves multiple third parties. Reusable Intellectual Property (IP) based SoC design methodology is cost effective, but it introduces trust and security concerns. A malicious third-party can insert hardware Trojans during any stages in the development cycle starting from design implementation to fabrication. These malicious modifications may alter the original functionality or leak secret information. To remain covert under in-field testing, a hardware Trojan is carefully designed to be triggered by extremely rare circuit input events. An example hardware Trojan is shown in Figure 1 with corresponding *trigger* and *payload*. The trigger condition is usually constructed by a few signals that can be activated under rare conditions. Figure 1 illustrates a beneficial way to assemble the rare signals (A, B and C) to form a rare input event. If the selected signals are independent, the probability of triggering this condition is multiplication of all the probabilities of these signals. Due to the stealthy nature of these Trojans, the trigger condition may not be activated during traditional validation and regression testing. Once the trigger condition is activated, the effects of the hardware Trojan can flip the value of the payload, which could be a signal to control the privilege, alter an important function or send secret information to the outside [1]. Therefore, it is paramount to have efficient validation approaches that can activate rare trigger conditions to enable Trojan detection.

Y. Lyu and P. Mishra are with the Department of Computer and Information Science and Engineering at the University of Florida, Gainesville, FL, 32611-6120, USA. (email: lvyangdi@ufl.edu).

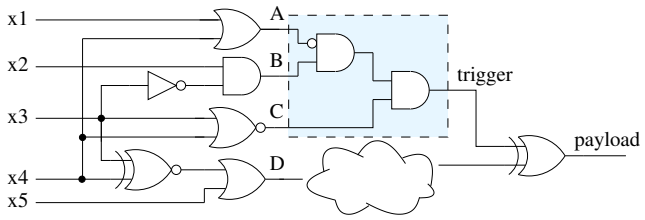


Fig. 1: An example hardware Trojan with a trigger condition constructed by three rare signals (A, B, C). An attacker can construct a valid trigger condition with $(A == 0 \wedge B == 1 \wedge C == 1)$. When this rare condition is satisfied, the value of payload is changed. The alteration of payload can introduce malfunction or information leakage.

To detect hardware Trojans, various approaches have been proposed including logic testing [2]–[7] and side-channel analysis [8]–[13]. However, existing approaches are neither effective nor scalable to large designs with extremely rare trigger conditions. Logic testing requires test vectors to fully activate trigger condition and also propagate the effects to observable outputs. In contrast, side-channel analysis detects hardware Trojans by observing the side effects of inserted gates. Since the Trojans are very small (a few gates in a million-gate design), their side-channel footprint can easily hide within process variation and environmental noise margins [14], [15]. Although side-channel analysis does not require activation of trigger conditions, the activation is likely to improve the sensitivity for several types of side-channel analysis, such as current switch-based [10], electromagnetic radiation based [16] and delay-based [8] side-channel analysis. Let us use the design in Figure 1 as an example to show the benefit of trigger activation in side-channel analysis based on dynamic current. When the trigger is not activated, the difference between the current switches from the golden design and the Trojan-inserted design is at most 5, which is the number of extra signals inserted by the Trojans. It is due to the fact that the only difference between the golden design and the Trojan-inserted design is the shaded area in Figure 1. However, when the trigger is activated, there would be more switching difference from the signals that are affected by the payload, which improves the side-channel sensitivity beyond the process variations and environmental noise. Therefore,

trigger activation is a fundamental problem in both logic testing and side-channel analysis based Trojan detection.

Trigger activation is a major challenge due to the exponentially large space that an adversary can exploit to construct trigger conditions. Conventional validation approaches simulate the design using millions or billions of random/constrained-random test vectors, and hope that one of these tests will activate the trigger condition. MERO [3] is one of the constrained-random test generation approaches that is similar to $N-$detect stuck-at ATPG [17], [18]. However, it is not effective in large designs with extremely rare trigger conditions as demonstrated in Section III. Existing directed test generation techniques are beneficial for known targets, but not useful for unknown targets (trigger conditions) since it leads to exponential complexity as discussed in Section II.

A. Threat Model

An attacker may insert hardware Trojans during any design stages, and use any signals as trigger points, including primary inputs and internal signals. In this paper, we generate tests by analyzing the gate-level design (netlist), which may or may not contain hardware Trojans. If the netlist that we analyze contains hardware Trojans, our goal is to activate these Trojans, which is similar to VeriTrust [5]. On the other hand, when the netlist is a golden design, our goal is to generate tests that will maximize the probability of activating the hardware Trojans that may be inserted in the future, which is similar to MERO [3]. Since the latter is a much harder problem, this paper will focus on unknown future Trojans except for the experiments in Section V-G. In the remainder of this paper, we will use design (netlist) to represent the golden netlist for test generation, and Design-Under-Test (DUT) to represent the Trojan-inserted design for evaluation. DUT can be a pre-silicon netlist or post-silicon integrated circuit.

B. Contributions

In this paper, we solve the trigger activation problem by mapping it to the problem of covering maximal cliques in a graph. Our goal is to activate extremely rare trigger conditions that can be covert during traditional validation. The major contributions of this paper are as follows:

1) To the best of our knowledge, our approach is the first attempt to map trigger activation problem to maximal clique cover problem. We prove that the test vectors generated by covering maximal cliques are complete and compact considering trigger coverage and test length.
2) We propose an efficient test generation algorithm for Trigger Activation by Repeated MAximal Clique sampling (TARMAC).
3) We outline an algorithm to support concurrent execution of time-consuming computations to improve the scalability of TARMAC.
4) Experimental results demonstrate that TARMAC outperforms the state-of-the-art test generation techniques by several orders-of-magnitude for extremely rare-to-activate trigger conditions in large designs.

The rest of this paper is organized as follows. Section II surveys prior efforts in trigger activation. In Section III, we motivate the need for this work by highlighting the drawbacks of $N-$detect paradigm as well as the limitations of the state-of-the-art test generation approaches. Section IV describes our proposed test generation framework. Section V presents the experimental results. Finally, Section VI concludes this paper.

II. Background and Related Work

In this section, we outline the related efforts in test generation and provide background on maximal clique problem.

A. Directed Test Generation

Random and constrained-random tests are widely used in traditional functional validation methodology. Unfortunately, even billions or trillions of constrained-random tests cannot cover many complex and corner-case scenarios in today's industrial designs. Directed tests are promising in such cases to activate the specific targets that were not covered by random or constrained-random tests. There are a wide variety of directed test generation techniques [5], [19]–[26] for functional validation. It is a major challenge to directly activate unknown Trojans due to exponential complexity. For example, even for a small ISCAS benchmark (c880 with only 451 gates) [27], there are approximately 10^{11} triggers possible with only four trigger points. The number would be exponentially higher if we consider triggers with different number of trigger points. Clearly, it is infeasible to generate and apply so many directed tests to activate Trojan triggers even for a tiny benchmark. Kitsos et al. [19] proposed a promising test generation approach to detect hardware Trojans based on combinatorial testing to significantly reduce the test sizes while providing mathematical guarantees for search space coverage. There are some recent efforts in Trojan detection using concolic testing [28], [29]. Unfortunately, these techniques are not beneficial for extremely rare trigger conditions since it leads to exponential complexity. Therefore, directed test generation is not useful for activating rare (and unknown) trigger conditions in large designs.

B. Statistical Test Generation

Statistical test generation is a promising alternative to directed tests. The basic idea is to activate the rare signals as much as possible (one or more at a time) to increase the likelihood of activating the actual (unknown) trigger consisting of rare signals. Extensive research has been done on statistical test generation combining ATPG and $N-$detect paradigm [3]. In [3], the authors proposed a tool named MERO to generate $N-$detect test for logic testing. Algorithm 1 shows the main steps of MERO. The goal of $N-$detect is to generate test vectors to activate each rare signal N times. MERO achieves $N-$detect criteria by constrained random approach. It starts with a large number of random test vectors, and flips each bit of random vectors to increase $N-$detect criteria. If a flip can increase the activation of rare signals which have not been activated by N times, the algorithm keeps the flipped

pattern (reverses the flipping otherwise). MERO is shown to be effective in small designs (e.g., ISCAS benchmarks [27], [30]) with relatively easy-to-activate trigger conditions (with four rare signals, and larger than 0.1 rareness threshold). However, MERO is unsuitable for large designs (scalability problem) as well as hard-to-detect triggers [7] as demonstrated in Section III.

Algorithm 1 MERO [3]

```
procedure MERO(R, N)
    Tests = {}
    simulate design with R random vectors
    sort random vectors by the number of rare signal hits
    for each vector u in random vectors do
        for each bit u_i in u do
            Flip u_i and simulate the design
            if N−detect criteria does not improve then
                reverse flipping
            end if
        end for
        Tests = Tests ∪ u, if u improves N-detect criteria
    end for
end procedure
```

C. Maximal Clique Problem

Clique decision problem is listed as one of Karp's 21 NP-complete problems [31]. Maximal clique problem [32] is the problem that given a set of vertices and their connectivity, find the maximal clique that no other vertex can be added. As proved by Moon and Moser [33], the number of maximal cliques is $O(3^{n/3})$ for n vertices in the worst case. Therefore, the effort of listing all maximal cliques is exponential to the number of vertices. Many efficient and parallel approaches [34], [35] exist in practice. BronKerbosch algorithm [34] is a widely used approach to list all maximal cliques in a graph. It is a recursive procedure that keeps track of three disjoint sets R, P and X, representing constructed clique, candidate vertices and excluded vertices, respectively. The existence of X ensures that maximal cliques are not repeated. Each recursive call adds one vertex from P to R and reports maximal clique when P and X are both empty. In this paper, we utilize maximal clique to solve the trigger activation problem as described in Section IV.

III. Motivation

$N-$detect paradigm has been successful in both logic testing [3], [18] and side-channel analysis [36]. $N-$detect paradigm requires the test set to activate each rare signal N times and is statistically effective for trigger activation given "sufficiently" large N [3]. The probability of activating trigger conditions will significantly decrease when the trigger condition is composed of very rare signals. It is expected that increasing N can increase the chances of hitting trigger conditions. However, larger N will significantly deteriorate the test generation performance and increase the required test length. MERO incorporated $N-$detect idea [3] with deterministic flipping method as shown in Algorithm 1, and the quality of generated test vectors is highly dependent on the quality of the initial random vectors. MERO has the following two major problems that make it ineffective for activating hard-to-detect trigger conditions in large designs.

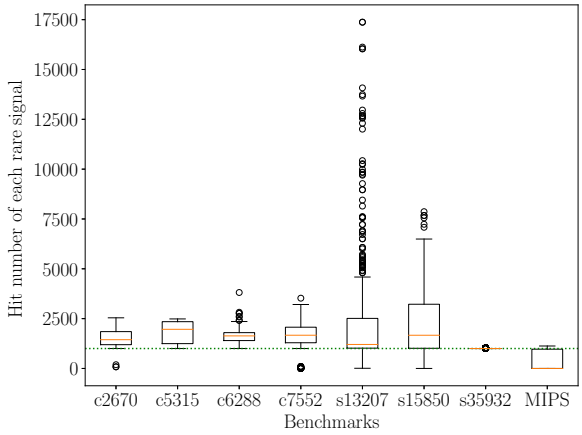


Fig. 2: The number of times each rare signal is activated by the test vectors generated by MERO for ISCAS benchmarks [27], [30] and MIPS processor [37]. The number of initial random vectors is 100K for ISCAS benchmarks and one million for MIPS processor. N is fixed to be 1000 for $N-$detect criteria (green line). Extremely rare signals are almost never activated while not-so-rare signals are activated more than N times.

Scalability Problem: Although MERO claimed to implement $N-$detect, the generated test vectors cannot guarantee that each rare signal is activated at least N times. With the same configuration ($R = 100K, N = 1000$) for the same ISCAS benchmarks [3] and ($R = 1M, N = 1000$) for MIPS processor from [37], we examined the number of times each rare signal is activated by MERO as shown in Figure 2. There are some extremely rare signals (outliers below the green line) that are almost never activated in most benchmarks, while some signals (outliers above the green line) are activated more than N times. To ensure $N-$detect for all rare signals, the number of initial random vectors should be extremely large even for small benchmarks. To show how the number of random vectors affects $N-$detect in MERO, we set $N = 1000$ and vary the number of random vectors. The percentage of rare signals that are activated more than 1000 times is shown in Figure 3. As expected, the percentage of $N-$detect rare signals grows rapidly when the number of random vectors is small, but very slowly beyond a specific number. It is expected that for large designs, billions of random vectors are required to satisfy $N = 1000$. MERO requires *one simulation per bit flipping*, the total number of simulations would be in the order of billions or trillions, which makes this approach impractical for large designs.

Poor Trigger Coverage: MERO uses a vague notion of N being "sufficiently" large to ensure high trigger coverage. In fact, MERO simply selected $N = 1000$ in [3] for all benchmarks. Despite the fact that all rare signals are activated at least 1000 times in the small benchmark, such as c5315,

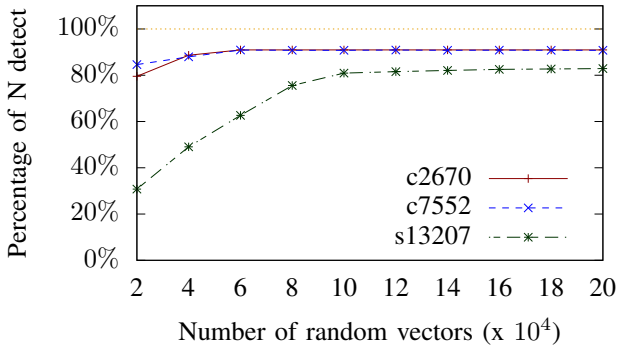


Fig. 3: The percentage of rare signals that are activated at least N times by MERO with the same configurations as Figure 2. The percentage of $N-$detect rare signals grows rapidly when the number of random vectors is small, but very slowly beyond a specific number of random vectors.

(see Figure 2), the trigger coverage is only 50.6% (see Section V-C). In other words, $N = 1000$ is not "sufficiently" large for such a small benchmark. For larger designs with more trigger points and lower rareness threshold, larger N is required to reach even a reasonable coverage by MERO, which needs drastically larger number of initial random vectors as discussed above, making scalability issue even worse.

Given the poor trigger coverage and scalability issue of MERO and $N-$detect, new paradigms are need to solve trigger activation problem. In this paper, we address the fundamental challenge of trigger activation by mapping it to clique cover problem and finding the test patterns to cover maximal cliques, as outlined in the next section.

IV. Scalable Activation of Rare Triggers

In this section, we propose a new test generation paradigm (TARMAC) to solve trigger activation problem by mapping it to maximal clique cover problem, as shown in Figure 4. Our approach first constructs a satisfiability graph based on the design (e.g., gate-level netlist). Next, it finds maximal satisfiable cliques ($\mathcal{MSC}$s) in the satisfiability graph. Finally, it utilizes a SAT solver [38] to generate one test for each maximal satisfiable clique. This section is organized as follows. We first define a few terms that are used in the paper. Next, we describe the mapping of trigger activation to clique cover problem and prove that the generated test set is complete and compact. Finally, we describe three test generation algorithms to find and cover maximal satisfiable cliques using directed clique enumeration (Algorithm 2), random sampling and lazy construction of satisfiability graph (Algorithm 3), and scalable TARMAC with multi-threaded execution (Algorithm 4), respectively.

A. Definition and Notations

Without any loss of generality, in this paper, we consider gate-level implementation of designs. We call the graph level representation of the design *a Design Graph* ($\mathcal{DG}$), where each vertex represents a signal and each edge represents the

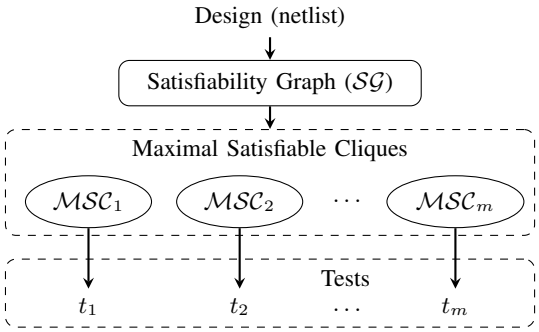


Fig. 4: Overview of our proposed (TARMAC) paradigm.

connectivity (via a gate). For each signal, we compute its logic expression (le) from its corresponding logic cone. For example, the logical expression of vertex A in Figure 1 is $A.le = x_1 \vee x_4$. For sequential circuits, we assume that design-for-debug architecture (e.g., scan chain) exists and the logic expression can be formulated using any register values.

We assume that a subset of signals and their rare values rv are given, from which the trigger conditions will be constructed. We refer to them as ***potential trigger signals (PTS)***. PTS could be any subset of signals, selected through static analysis or random simulation. Their rare values can be arbitrarily chosen by the designers. For example, [3] assumed that Trojans can only be constructed from rare signals with their less satisfied values which are identified using random simulation. A trigger signal is *activated* if it satisfies its rare value. We define *satisfiability graph* as follows.

Definition 1. *A* **Satisfiability Graph** *($\mathcal{SG}$) consists of vertices representing PTS and their satisfiability connections, $\mathcal{SG} = \{\mathcal{V}, \mathcal{E}\}$ where $\mathcal{V} == PTS$. If $(u.le == u.rv) \wedge (v.le == v.rv)$ is satisfiable, then there exists an edge between u and v, i.e., $u \in \mathcal{E}(v)$ and $v \in \mathcal{E}(u)$.*

Let us consider the example in Figure 1 with four PTS (A, B, C, D) and their corresponding rare values (0, 1, 1, 0). To construct the satisfiability graph for this example, we need to use their logical expressions described above and determine their connectivity. To find out if there is an edge between any two vertices, we check if any input (test) pattern exists that satisfies both rare values. For example, the edge between A and B exists since input pattern 01000 satisfies the condition $(x_1 \vee x_4 == 0) \wedge (x_2 \wedge \neg x_3 == 1)$. In other words, 01000 can activate both A and B at the same time with their respective rare values. On the other hand, there is no input pattern that satisfies $(\neg(x_3 \vee x_4) == 1) \wedge (\neg(x_3 \oplus x_4) \vee x_5 == 0)$, i.e., there is no edge between C and D. The constructed satisfiability graph is shown in Figure 5 (logic expressions and rare values are shown inside parentheses). It is easy to see that $\mathcal{SG}$ is an undirected graph.

B. Mapping Trigger Activation to Clique Cover Problem

A fundamental contribution of this paper is to show that trigger activation problem can be mapped to clique cover problem. First, we show that any valid trigger condition forms a clique in satisfiability graph $\mathcal{SG}$.

Lemma 1. *For any valid trigger condition with k rare signals $\{v_1, v_2, ..., v_k\}$, the vertices $\{v_1, v_2, ..., v_k\}$ form a $k-$clique in the satisfiability graph $\mathcal{SG}$.*

Proof. We prove Lemma 1 by contradiction. Assume that there is no edge between v_i and v_j. By definition, condition $(v_i.le == v_i.rv) \wedge (v_j.le == v_j.rv)$ is not satisfiable. Therefore, there will be no test that can activate v_i and v_j together, invalidating the trigger condition. Since there is an edge between any pair of vertices, $\{v_1, v_2, ..., v_k\}$ form a $k-$clique in the satisfiability graph $\mathcal{SG}$. □

Note that it is possible to have a clique in the satisfiability graph that does not represent a valid trigger condition. For example, consider the clique ABD in Figure 5. There is no input pattern that satisfies the condition $(x_1 \vee x_4 == 0) \wedge (x_2 \wedge \neg x_3 == 1) \wedge (\neg(x_3 \oplus x_4) \vee x_5 == 0)$, although there are edges between any two of the three vertices. In other words, ABD forms a clique in $\mathcal{SG}$, but it does not represent a valid trigger condition. Clearly, an adversary will not use it as a Trojan trigger since it cannot be triggered. For the ease of illustration, we define *satisfiable clique* in Definition 2. The relationship between satisfiable cliques and valid trigger conditions is shown in Lemma 2 and Lemma 3.

Definition 2. *A satisfiable clique $\mathcal{SC}$ is a clique in a satisfiability graph $\mathcal{SG}$, where all the vertices of $\mathcal{SC}$ can be activated by the same input vector.*

Lemma 2. *Any valid trigger condition can be represented as a satisfiable clique $\mathcal{SC}$ in satisfiability graph $\mathcal{SG}$.*

Proof. Lemma 1 proves that any valid trigger condition forms a clique in $\mathcal{SG}$. Validity of this trigger condition ensures that all vertices can be activated by the same input vector. By Definition 2, this clique is a satisfiable clique. □

Lemma 3. *Any satisfiable clique $\mathcal{SC}$ in satisfiability graph $\mathcal{SG}$ represents a valid trigger condition.*

Proof. For any satisfiable clique, all its vertices can be activated by a test vector by Definition 2. Thus, these vertices represent a valid trigger condition. □

Finally, we explore the mapping from the set of valid trigger conditions to the set of satisfiable cliques in Theorem 1. It points out a new way to solve trigger activation problem, i.e., finding test vectors to cover satisfiable cliques in a satisfiability graph.

Theorem 1. *The mapping between the set of valid trigger conditions and the set of satisfiable cliques is a bijection.*

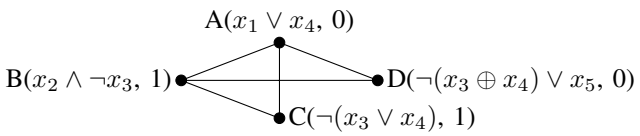


Fig. 5: Satisfiability graph with 4 PTS (A,B,C,D) from Figure 1, with logic expressions and rare values in parentheses.

Proof. As different trigger conditions consist of at least one different rare signal, the corresponding satisfiable cliques have at least one different vertex. Hence, no two valid trigger conditions map to the same satisfiable clique, i.e., the mapping from the set of valid trigger conditions to the set of satisfiable cliques is an injection from Lemma 2. Similarly, we can prove that the mapping from the set of satisfiable cliques to the set of valid trigger conditions is also an injection from Lemma 3. Therefore, we have a one-to-one mapping between these two sets. □

C. Directed Test Generation Scheme

Lemma 4. *If one test vector can satisfy a satisfiable clique, all its subgraphs can be satisfied by the same test vector.*

Proof. Let $\mathcal{R}$ be a subgraph of a satisfiable clique $\mathcal{SC}$. By Definition 2, all vertices in $\mathcal{SC}$ can be satisfied by the same test vector t. All vertices of $\mathcal{R}$ are inherently satisfiable by t since the vertices of $\mathcal{R}$ is a subset of the vertices of $\mathcal{SC}$. □

Lemma 5. *A subgraph of a satisfiable clique is also a satisfiable clique.*

Proof. For any satisfiable clique $\mathcal{SC}$, its subgraph $\mathcal{R}$ is a clique as $\mathcal{SC}$ is a clique. By Lemma 4, $\mathcal{R}$ is satisfiable. By definition, $\mathcal{R}$ is a satisfiable clique. □

Therefore, if we are able to find a test vector that can satisfy a clique, it is not necessary to generate any more test for all the trigger conditions represented by its subgraphs. Clearly, *the most profitable test vector is the one that can satisfy the largest clique*. Similar to cliques in graph theory, we define a *maximal satisfiable clique* in Definition 3.

Definition 3. *A maximal satisfiable clique ($\mathcal{MSC}$) is a satisfiable clique to which no more vertices can be added.*

Let $\{\mathcal{MSC}_1, \mathcal{MSC}_2, \ldots, \mathcal{MSC}_n\}$ represents the complete set of maximal satisfiable cliques, where n is the total number of maximal satisfiable cliques. For example, $\{\mathcal{MSC}_1$ = ABC, $\mathcal{MSC}_2$ = AD, $\mathcal{MSC}_3$ = BD$\}$ represents the complete set of maximal satisfiable cliques in Figure 5. Next, we prove that the set of test vectors that activate all elements in $\{\mathcal{MSC}_i\}$ is optimal in activating all possible trigger conditions in the design.

Theorem 2. *Let t_i be an input pattern that activates the corresponding maximal satisfiable clique $\mathcal{MSC}_i$. Then, the test set $T = \{t_i\}$ is complete and compact, i.e., it is the shortest test set that can activate all valid trigger conditions.*

Proof. We first prove the completeness of our test set. For any valid trigger condition, it forms a satisfiable clique $\mathcal{SC}$ by Theorem 1. By definition of maximal satisfiable cliques, there exists some maximal satisfiable clique $\mathcal{MSC}_i$ such that $\mathcal{SC} \subset \mathcal{MSC}_i$. As $t_i \in T$ satisfies $\mathcal{MSC}_i$, it inherently satisfies satisfiable clique $\mathcal{SC}$ by Lemma 4. As T can satisfy all elements in $\{\mathcal{MSC}_i\}$, it can satisfy any valid trigger condition.

Now, we prove that the test set is compact. It is easy to see that any two maximal satisfiable cliques can never be activated by the same test vector, otherwise, they form a larger

satisfiable clique which contradicts the definition of maximal satisfiable clique in Definition 3. As any maximal satisfiable clique represents a valid trigger condition by Lemma 3, a test set that can activate all these trigger conditions need at least $|\{\mathcal{MSC}_i\}|(=|T|)$ test vectors. Hence, no test set that satisfies all trigger conditions can be shorter than T. □

As a result, the problem of test generation for trigger activation can be reduced and mapped to the problem of finding maximal satisfiable cliques and generate directed test for them. Based on Theorem 2, the generated test vectors are the optimal solution considering both trigger coverage and test length. For the example in Figure 5, we need exactly three tests - t_1 (01000), t_2 (01100) and t_3 (11010) to activate maximal satisfiability cliques ABC, AD, and BD, respectively.

D. Test Generation Algorithms

In this section, we present two test generation algorithms to generate test patterns by covering maximal satisfiability cliques. Algorithm 2 (Section IV-D1) is guaranteed to generate the complete test set (covers all the trigger conditions) but is not scalable since it requires enumeration of potentially exponential number of $\mathcal{MSC}s$. In addition, it has the bottleneck of construction of the full satisfiability graph. This algorithm is suitable when only a small number of rare signals are in a design. To address the scalability issue, Algorithm 3 (Section IV-D2) replaces the enumeration problem by randomly sampling $\mathcal{MSC}s$, and it performs lazy construction of the satisfiability graph. It is significantly faster and effective, but cannot guarantee completeness. The reminder of this section describes these algorithms.

1) Test Generation using Clique Enumeration: Based on Theorem 2, we propose our first straightforward test generation algorithm based on *clique enumeration*. The main steps of this approach are shown in Algorithm 2. The procedure of $TestGeneration$ first parses and constructs the design graph ($\mathcal{DG}$) from the gate-level netlist, and computes all the logic expressions. Then, the vertices of satisifiability graph ($\mathcal{SG}$) are initialized from PTS and the edges are constructed after testing satisfiability of any two vertices ($ConstructSatisfiabilityGraph$). Next, Bron-Kerbosch algorithm [34] is applied to find all maximal cliques in $\mathcal{SG}$. For every clique $\mathcal{C}$ found in line 6, we need to find all maximal satisfiable cliques inside $\mathcal{C}$. Finally, test vectors are generated for each maximal satisfiable clique.

Next, we prove that the generated test vectors are complete. For any maximal satisfiable clique, it must be a subgraph of some maximal clique $\mathcal{C}$ enumerated by Bron-Kerbisch [34]. Line 7 ensures that all maximal satisfiable cliques are found when we visit $\mathcal{C}$. By Theorem 2, the generated test vectors are complete.

This approach is effective in small designs, but it lacks the scalability due to the following three major bottlenecks:

- The computational problem of finding all maximal cliques is NP-hard. Although BronKerbosch algorithm [34] is practical in finding all maximal cliques, it suffers from deep recursive function calls for large graphs with the worst running time $O(3^{n/3})$, where n is the number of vertices.
- Finding all maximal satisfiable cliques inside a large clique (e.g., more than 20 vertices) is difficult. A brute-force approach need to check the satisfiability of all possible combinations. The running time is exponential to the size of the clique.
- Algorithm 2 also has the bottleneck of constructing the full satisfiability graph. When the number of vertices $|\mathcal{SG}.\mathcal{V}|$ is extremely large, checking if an edge exists between two vertices requires approximately $|\mathcal{SG}.\mathcal{V}|^2/2$ calls of the SMT solver, which can be prohibitive in terms of debug time.

Algorithm 2 Test Generation by Clique Enumeration

1: **procedure** $TestGeneration$(circuit netlist CN, potential trigger signals PTS)
2: $\mathcal{DG}$ = ConstructDesignGraph (CN)
3: Compute logic expressions for PTS in $\mathcal{DG}$
4: $\mathcal{SG}$ = ConstructSatisfiabilityGraph($\mathcal{DG}$, PTS)
5: Clique set $\mathcal{CS}$ = Bron-Kerbosch($\mathcal{SG}$)
6: **for** each clique $\mathcal{C}$ in $\mathcal{CS}$ **do**
7: **for** each maximal satisfiable clique in $\mathcal{C}$ **do**
8: Use SMT solver to generate a test vector t_i for it
9: **end for**
10: **end for**
11: *return* $Tests = \{t_1, t_2, \ldots, t_n\}$
12: **end procedure**

13: **procedure** $ConstructSatisfiabilityGraph$($\mathcal{DG}$, PTS)
14: $\mathcal{SG}.\mathcal{V}$ = PTS, $\mathcal{SG}.\mathcal{E}(u) = \{\}$
15: **for** $u, v \in \mathcal{SG}.\mathcal{V}$ **do**
16: SAT expression S = $(u.le == u.rv) \wedge (v.le == v.rv)$
17: **if** satisfiabile(S) **then**
18: $\mathcal{SG}.\mathcal{E}(v) = \mathcal{SG}.\mathcal{E}(v) \cup \{u\}$
19: $\mathcal{SG}.\mathcal{E}(u) = \mathcal{SG}.\mathcal{E}(u) \cup \{v\}$
20: **end if**
21: **end for**
22: *return* $\mathcal{SG}$
23: **end procedure**

2) Efficient Test Generation using Clique Sampling and Lazy Construction: To address both clique enumeration and satisfiability graph construction issues in Algorithm 2, we propose an on-the-fly technique (TARMAC) in Algorithm 3 that utilizes lazy construction of the satisfiability graph and random sampling of maximal satisfiable cliques. The random sampling makes TARMAC scalable to large designs with the cost of completeness. For each sampled maximal satisfiable clique, TARMAC generates one test vector for it.

Clique sampling is done by maintaining two sets of variables: cns to keep track of constraints that are satisfiable (represents vertices that are already found in a satisfiable clique), and P to represent candidate vertices that may potentially be added to the clique. Initially, cns is true and P contains all the vertices. We first randomly select and remove a vertex v from

candidate set P. If cns can be augmented by $v.le == v.rv$, we put it into cns and remove all vertices in P that are not connected to v (line 16). It is easy to verify that cns represents a maximal satisfiable clique when P is empty. Parameter *VN* is used to control how many times we should sample maximal satisfiable cliques, i.e., the number of generated test vectors.

The complexity of full satisfiability graph construction is eliminated by lazy construction. As shown in Algorithm 3, initially every vertex is connected to every other vertices in line 3. Whenever we find two vertices unsatisfiable (line 17), we remove the edge between these two vertices. Lazy construction benefits large designs by generating test vectors as soon as possible, with the cost of wasted SMT solver calls. If we look at the example in Figure 5, Algorithm 2 disconnects C and D before searching for cliques, while Algorithm 3 constructs a fully connected graph initially, which may introduce multiple wasted SMT solver calls in the clique sampling process involving C and D. These two vertices will be disconnected in line 17-19 only when they are selected as the first two vertices from P in line 13, with the probability of approximately $2/|\mathcal{SG}.\mathcal{V}|^2$. Statistically, the full graph will be constructed after $|\mathcal{SG}.\mathcal{V}|^2/2$ sampling.

Algorithm 3 Test Generation using Random Sampling and Lazy Construction (TARMAC)

1: **procedure** $TARMAC$(circuit netlist CN, potential trigger signals PTS, maxVectorNumber VN)
2: $\mathcal{DG}$ = ConstructDesignGraph (CN)
3: Compute logic expressions for PTS in $\mathcal{DG}$
4: $\mathcal{SG}.\mathcal{V} = PTS$, $\mathcal{SG}.\mathcal{E}(u) = \mathcal{SG}.\mathcal{V} \setminus \{u\}$
5: **for** $i = 1$ to VN **do**
6: t_i = CliqueSampling($\mathcal{SG}$)
7: **end for**
8: *return* $Tests = \{t_1, t_2, \ldots, t_{VN}\}$
9: **end procedure**

10: **procedure** $CliqueSampling(\mathcal{SG})$
11: constraints $cns = true$, P = $\mathcal{SG}.\mathcal{V}$
12: **while** P is not empty **do**
13: randomly pick and remove a vertex v from P
14: **if** satisfiabile($cns \wedge (v.le == v.rv)$) **then**
15: $cns = cns \wedge (v.le == v.rv)$
16: $P = P \cap \mathcal{SG}.\mathcal{E}(v)$
17: **else if** cns has one constraint $u.le == u.rv$ **then**
18: $\mathcal{SG}.\mathcal{E}(v) = \mathcal{SG}.\mathcal{E}(v) \backslash \{u\}$
19: $\mathcal{SG}.\mathcal{E}(u) = \mathcal{SG}.\mathcal{E}(u) \backslash \{v\}$
20: **end if**
21: **end while**
22: Use SMT solver to solve cns and *return* the test
23: **end procedure**

E. Scalable TRAMAC by Parallelization of Clique Sampling

By inspecting the process of clique sampling, we can see that this process is highly parallelizable. To further increase the efficiency of Algorithm 3, we add parallelism to clique sampling, i.e., $TARMAC_p$, as shown in Algorithm 4. Instead of generating all VN test vectors in one thread, $TARMAC_p$ evenly splits the task into NT threads, where each thread generates a batch of $VN_p = VN/NT$ test vectors. In order to minimize the overlapped efforts of covering the same cliques by different threads, we feed a different random seed to each batch sampling (line 7 and 8). Then, each thread runs $batchSampling$ independently. It sets the random seed, and calling the modified version of clique sampling to generate VN_p test vectors. After a thread completes its job, the generated test vectors are appended to the list of final tests. Comparing the clique sampling in Algorithm 4 and Algorithm 3, the only differences are line 28 and 31, where mutex is used to safely update the edges of shared satisfiability graph $\mathcal{SG}.\mathcal{E}$. Except for this block, the data structures are either copied, e.g. $\mathcal{SG}.\mathcal{V}$ in line 21, or are only for reading, e.g., $\mathcal{SG}.\mathcal{E}$ in line 26. For efficiency consideration, a simple mutex is used to prevent multiple writing to $\mathcal{SG}.\mathcal{E}$, instead of a readers-writer lock. In other words, this simple mechanism allows multiple threads to read $\mathcal{SG}.\mathcal{E}$ (line 26) while one thread is writing to it. The only difference compared to a readers-writer lock is that simple mutex mechanism will read old version of $\mathcal{SG}.\mathcal{E}$ in line 26, which makes P to contain one redundant vertex. It is not critical since the redundant vertex will be removed in a future iteration anyway. When multi-core infrastructures are provided, $TARMAC_p$ can achieve high efficiency improvement over $TARMAC$ due to the parallelism of constraints solving.

F. Effectiveness of Random Clique Sampling

In Section IV-D, we introduced two algorithms, i.e., clique enumeration (Algorithm 2), and random clique sampling with lazy construction (TARMAC, Algorithm 3). As expected, random sampling cannot guarantee to find all maximal satisfiable cliques as clique enumeration. In this section, we show why random sampling is still effective.

Let us consider two scenarios shown in Figure 6, where the circles of C_1, C_2, C_3 represent maximal SAT cliques, and the octagon represents the 8-trigger condition. The only difference between Figure 6(a) and Figure 6(b) is the size of maximal SAT cliques. In the large clique scenario (Figure 6(a)), the average size of maximal SAT cliques is 200, while the average size is 20 in the small clique scenario (Figure 6(b)). In the large clique scenario, the 8-trigger condition is more likely to be in the overlap areas of many maximal satisfiable cliques as shown in Figure 6(a). In this case, random sampling (Algorithm 3) can easily activate the trigger condition by generating a test vector to cover any of the maximal satisfiable cliques that are a super set of the trigger condition. On the other hand, the size of the 8-trigger condition is close to the average maximal clique size in the small clique scenario. As a result, it is less likely to be activated by random sampling since it is covered by a small number of maximal satisfiable cliques. In the extreme case, e.g., the size of trigger condition is the same as the size of maximal SAT cliques, we need to enumerate all maximal SAT cliques as Algorithm 2. In fact, it is the best any test generation approach for this case. In most of the benchmarks, we observe a relatively large maximal satisfiable cliques compared to trigger points, as shown in Section V-F.

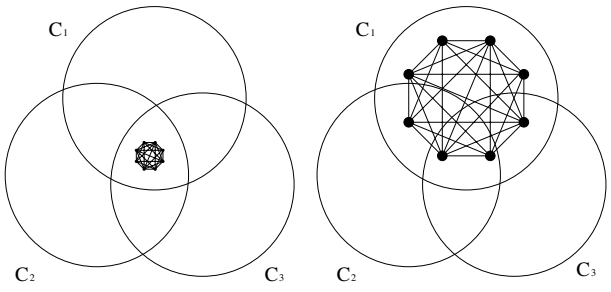
Algorithm 4 Parallelization of TARMAC

procedure $TARMAC_p$(circuit netlist CN, potential trigger signals PTS, maxVectorNumber VN, number of threads NT)
  $\mathcal{DG}$ = ConstructDesignGraph (VN)
  Compute logic expressions for PTS in $\mathcal{DG}$
  $\mathcal{SG}.\mathcal{V} = PTS$, $\mathcal{SG}.\mathcal{E}(u) = \mathcal{SG}.\mathcal{V} \setminus \{u\}$
  The number of vectors per thread $VN_p = VN/NT$
  for $td = 1$ to NT **do** // NT threads
    $seed = random()$
    Create a new thread to execute $batchSampling$ ($\mathcal{SG}$, $VN_p, seed$)
    Append the generated tests to $Tests$
  end for
  return $Tests = \{t_1, t_2, \ldots, t_{VN}\}$
end procedure

procedure $batchSampling(\mathcal{SG}, VN_p, seed)$
  $random.seed(seed)$
  for $i = 1$ to VN_p **do**
    t_i = CliqueSampling($\mathcal{SG}$)
  end for
  return batchTests $= \{t_1, t_2, \ldots, t_{VN_p}\}$
end procedure

procedure $CliqueSampling(\mathcal{SG})$
  constraints $cns = true$, P = $\mathcal{SG}.\mathcal{V}$
  while P is not empty **do**
    randomly pick and remove a vertex v from P
    if satisfiabile($cns \wedge (v.le == v.rv)$) **then**
      $cns = cns \wedge (v.le == v.rv)$
      $P = P \cap \mathcal{SG}.\mathcal{E}(v)$
    else if cns has one constraint $u.le == u.rv$ **then**
      $mutex.lock()$ // Protect shared graphs
      $\mathcal{SG}.\mathcal{E}(v) = \mathcal{SG}.\mathcal{E}(v) \backslash \{u\}$
      $\mathcal{SG}.\mathcal{E}(u) = \mathcal{SG}.\mathcal{E}(u) \backslash \{v\}$
      $mutex.unlock()$
    end if
  end while
  Use SMT solver to solve cns and *return* the test
end procedure

In summary, our paradigm reduced and mapped the problem of trigger activation to the problem of covering maximal satisfiable cliques. The choice between clique enumeration and random sampling is based on the relative size of maximal satisfiable cliques and the trigger points. When an adversary is allowed to construct any size of trigger condition, e.g., a size close to the maximal SAT cliques, Algorithm 2 is the optimum way to generate tests. However, it is not realistic in practice. An adversary tends to select a small number of trigger points considering area and power constraints in the design and to bypass side-channel analysis. In this scenario, random sampling (Algorithm 3) further reduces the problem size by selecting the representative maximal satisfiable cliques. As shown in the above example, each 8-trigger condition could possibly be covered by a large number of maximal satisfiable cliques of average size 200. If one of them is sampled by our algorithm, the trigger activation problem is solved. It also points out an interesting direction to improve TARMAC. Instead of randomly sampling each time, a biased sampling technique could be beneficial to instruct the sampling process to cover cliques that have less overlap with already covered ones. In order to improve the performance further, we have modified TARMAC such that multiple threads can perform clique sampling in parallel. This will enable an efficient and scalable test generation framework for activating rare triggers.



(a) Large clique scenario (b) Small clique scenario

Fig. 6: The relative size of trigger conditions compared to maximal SAT cliques. (a) The average size of maximal SAT cliques is 200, and 8-trigger condition is relatively small which could possibly be covered by a large number of cliques. (b) The average size of maximal SAT cliques is 20, and 8-trigger condition is relatively large which is less likely to be covered by multiple cliques.

V. Experiments

A. Experimental Setup

The TARMAC framework is implemented in C++ with Z3 [38] as our SMT solver. This framework first parses gate-level Verilog files into design graphs ($\mathcal{DG}$). Then, for each signal in PTS, we utilizes Z3 C++ API to compute its logic expression. For sequential circuits, all registers are treated as free variables with the assumption of full scan mode. Next, this framework constructs a satisfiability graph ($\mathcal{SG}$) and continuously samples maximal satisfiability cliques ($\mathcal{MSC}$) as shown in Algorithm 3. For each sampled $\mathcal{MSC}$, function call to Z3 is used to produce a test. For multithreading (TARMAC$_p$) in Algorithm 4, C++ pthread library is used to create different threads.

We conducted a wide variety of experiments on a server with Intel Xeon E5-2698 CPU @2.20GHz and 528GB RAM to evaluate the performance of TARMAC compared to random test vectors and $N-$detect approach (MERO [3]). In this paper, we used the same benchmarks (ISCAS-85 [27] and ISCAS-89 [30]) from [3] to enable a fair comparison with MERO. We have also used two large designs (memory controller from TrustHub [39] and MIPS processor from OpenCores [37], MEM and MIPS for short) to demonstrate the scalability of our approach. The experimental setup is shown in Figure 7. We first ran a number of random simulations (100K for ISCAS and

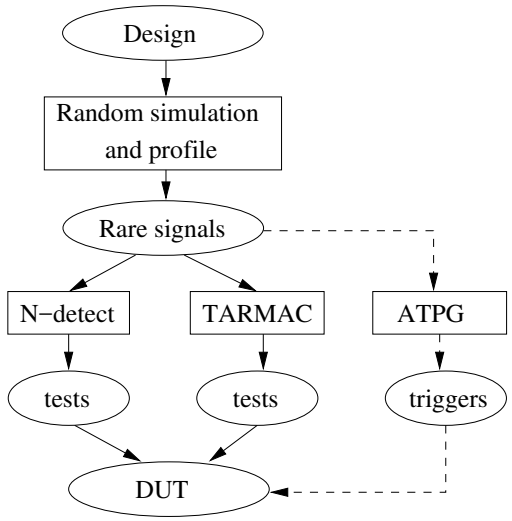


Fig. 7: Experimental setup for evaluation of TARMAC compared to $N-$detect approach. Trigger conditions are randomly sampled and validated by ATPG tools. Each DUT contains only one trigger condition. Test vectors from $N-$detect and TARMAC are applied to each DUT individually to collect trigger condition coverage information.

one million for MEM and MIPS) on the golden design[1] and computed the probability of each signal. Rareness threshold is set to 0.1 for ISCAS benchmarks and 0.005 for the other designs. We chose these rare signals as potential trigger signals (PTS). The rareness threshold is set in a way such that the constructed Trojans are likely to stay stealthy under both random/constrained random simulation and directed test generation. On one hand, the rareness threshold should be as small as possible, such that random/constrained random simulation is almost impossible to activate the constructed Trojans. On the other hand, the rareness threshold should also be large enough to allow as many choices of Trojans as possible, such that it is impossible to use directed test generation to cover all possibilities. We set the rareness threshold to be less than 0.1, and try to have around 1000 rare signals if possible, as shown in Table I. Theoretically, directed test generation need to try $\binom{1000}{8} \sim 2.4 \times 10^{19}$ times to cover all possible choices of 8-trigger Trojans with 1000 PTS.

For each benchmark, 1000 trigger conditions were randomly sampled and validated using ATPG. After sampling 1000 valid trigger conditions, each of them was individually integrated into the original design to construct a design under test (DUT). In other words, there are 1000 DUTs from each benchmark with one trigger condition for evaluation. We applied $N-$detect approach (MERO [3]), TARMAC (Algorithm 3) and TARMAC$_p$ (Algorithm 4) to generate the test sets. Finally, we applied test sets to each DUT and collected trigger condition coverage. For all experiments, we fixed $N = 1000$ for $N-$detect approaches.

[1]Except that we applied TARMAC directly on Trojan-inserted netlists in the DUT-known scenario in Section V-G, i.e., random simulation and rare signals computation are performed directly on Trojan-inserted netlists.

B. The Effects of Trigger Points

In the first experiment, we wanted to explore the effects of trigger points on the trigger coverage of MERO and TARMAC. When a trigger condition has less trigger points (e.g., 4), it has higher probability to be activated by random simulation. One the other hand, a trigger condition with more rare signals is much harder to activate. For example, the probability of activating a 16-trigger condition is less than 10^{-16} when these signals are independent and rareness threshold is 0.1.

We evaluated both MERO and TARMAC on c2670 and MIPS, with various number of trigger points between 4 and 16. The results of TARMAC and MERO are shown in Figure 8. Each line represents trigger condition coverage with respect to the number of test vectors applied to DUTs with a fixed number of trigger points. As the results suggest, the performance of MERO deteriorated sharply with increasing trigger points, while TARMAC maintained high coverage for both benchmarks. For small number of trigger points (e.g., 4), MERO can achieve good coverage in c2670. However, its coverage for large number of trigger points (e.g., 16) is extremely poor with less than 5% coverage. On the other hand, TARMAC can achieve 100% coverage with less than 100 test vectors even for 16-trigger conditions. As 16-trigger condition is more rare than 4-trigger ones, TARMAC took more test vectors to achieve the same coverage in MIPS as shown in Figure 8(b). Therefore, TARMAC is more resilient to the increasing number of trigger points and good at activating extremely rare-to-activate trigger conditions. In the remaining experiments, we fix the number of trigger points to be 8 since it is a common number of trigger points in TrustHub [39] and it allows MERO to achieve a reasonable trigger condition coverage for comparison.

C. Performance Evaluation

In this experiment, we compared the trigger condition coverage of TARMAC to random approach and MERO over all benchmarks. To get a fair comparison of trigger coverage, we evaluated the trigger coverage with the same number of test vectors. Note that the length of MERO test vectors cannot be controlled arbitrarily since it depends on the $N-$detect criteria and the number of initial random vectors R. Hence, we first ran MERO with ($R = 100,000, N = 1000$) for ISCAS benchmarks as suggested in [3] and ($R = 100,000, N = 1000$) for two large benchmarks. After MERO finished, we ran TARMAC to generate the same number of test vectors as MERO for each benchmark. The trigger coverage comparison of TARMAC with random and MERO test vectors is shown in Table I.

From Table I, we can see that TARMAC can achieve several orders-of-magnitude trigger coverage improvement over random test vectors in ISCAS benchmarks. TARMAC can provide and up to 49 times improvement in trigger coverage over MERO with four times reduction for generation of the same number of test vectors in the ISCAS benchmarks. For most benchmarks, TARMAC covered over 90% of the trigger conditions, while random and MERO test vectors missed most of them. In small benchmarks, such as c2670, c5315

TABLE I: Comparison of TARMAC with random simulation and MERO for trigger activation coverage over 1000 randomly sampled 8-trigger conditions. The test length of TARMAC is the same as MERO.

Bench	Number of rare signals	Random		MERO [3]			TARMAC					TARMAC$_p$ (64 cores)		
		Test Length	Cov. (%)	Test Length	Cov. (%)	Time (s)	Test Length	Cov. (%)	Impro. / Random	Impro. / MERO	Time (s)	Time (s)	Impro. / MERO	Impro. / TARMAC
c2670	43	100K	0.3	6820	38.2	1268	6820	100	333x	2.6x	257	4.3	295x	59.7x
c5315	164	100K	1.1	9232	50.6	4396	9232	98.8	89.8x	1.9x	682	13.3	330x	51.3x
c6288	169	100K	18.9	5044	76.6	596	5044	95.0	5.0x	1.2x	638	10.0	60x	63.8x
c7552	278	100K	0	14914	5.6	7871	14914	66.5	∞	11.9x	2185	41.6	189x	52.5x
s13207	604	100K	0	44534	1.9	15047	44534	94.4	∞	49.7x	5417	105.3	143x	51.4x
s15850	649	100K	0	39101	3	17000	39101	88.7	∞	29.6x	11337	205.4	83x	55.2x
s35932	1152	100K	100	4047	100	49616	4047	100	1x	1x	1947	38.9	1275x	50.1x
MEM	1306	1M	0	28542	0	89747	28542	98.6	∞	∞	15753	330.5	271x	47.7x
MIPS	906	1M	0	25042	0.2	273807	25042	95.6	∞	472x	19458	391.9	699x	49.7x
avg.	**586**	**300K**	**13.4**	**19697**	**30.7**	**51039**	**19697**	**93.1**	**$>$ 107x**	**$>$71x**	**6408**	**126.8**	**402x**	**50.5x**

TABLE II: Comparison of TARMAC with random simulation and MERO for trigger activation coverage over 1000 randomly sampled 8-trigger conditions. TARMAC is terminated when it just surpassed the same trigger coverage as MERO.

Bench	MERO			TARMAC				
	Test Length	Cov. (%)	Time (s)	Test Length	Reduction	Cov. (%)	Time (s)	Improvement
c2670	6820	38.2	1268	1	6820x	51.4	0.05	25360x
c5315	9232	50.6	4396	217	42.5x	50.6	19.1	230x
c6288	5044	76.6	596	284	17.8x	76.6	34.8	17x
c7552	14914	5.6	7871	175	85.2x	5.6	31.2	252x
s13207	44534	1.9	15047	5	8907x	2.6	0.8	18809x
s15850	39101	3	17000	13	3008x	3.3	4.3	3953x
MEM	28542	0	89747	1	28542x	1.9	1.1	81588x
MIPS	25042	0.2	273807	1	25042x	0.8	1.8	152115x
avg.	**21653**	**22.0**	**51216**	**87**	**249x**	**24.1**	**11.6**	**4415x**

and c6288, MERO outperformed random test vectors and achieved reasonable trigger condition coverage. However, in large benchmarks such as c7552, s13207 and s15850, the performance of MERO is very poor, with less than 6% trigger coverage. TARMAC, on the other hand, outperformed MERO in all ISCAS benchmarks with 91.9% trigger coverage on average. With the same number of test vectors, TARMAC can cover the extremely hard-to-activate trigger conditions that are left after applying both random test vectors and MERO with significantly less effort.

It is interesting to find that all three approaches did a great job in covering all trigger conditions in s35932. One of the reasons is that a lot of rare signals in s35932 can be satisfied together as shown in Section V-F. Another observation is that the quality of MERO is partially dependent on the quality of random test vectors. For example, with 18.9% and 100% trigger activation coverage from random test vectors for c6288 and s35932, respectively, test vectors from MERO can cover 76.6% and 100%. However, for benchmarks such as c7552 and s31207, test vectors of MERO can only achieve trigger coverage of 5.6% and 1.9%, respectively, since random test vectors cannot cover any trigger conditions. The limited improvement from random test vectors to MERO is due to the simple bit flipping to search for good vectors in MERO.

For the two large benchmark, MEM and MIPS, the number of rare signals are in the order of 1000. Since each trigger condition contains 8 rare signals with rareness threshold of 0.005, the probability of trigger conditions is less than 10^{-18}. It is expected that one million random simulations could not achieve good coverage. The test vectors generated by MERO also achieved poor coverage, 0% in memory controller, and 0.2% in MIPS. On the other hand, TARMAC is able to cover majority of the trigger conditions efficiently. For example, TARMAC covered 95.6% of trigger conditions in MIPS using the same amount of test vectors as MERO, but finished generation in 6 hours. Note that the average test generation of TARMAC for one test vector is less than one second. This demonstrates that TARMAC is scalable for large designs, while MERO is not suitable for large designs.

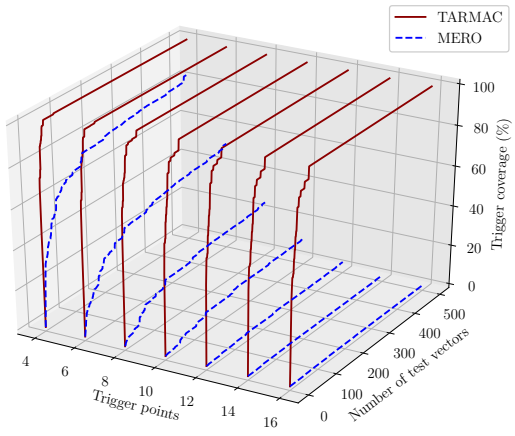
Overall, TARMAC provides drastic improvement in both trigger coverage (more than 107x and 71x over random test vectors and MERO, respectively) and test generation time (8x).

D. Parallelism Evaluation

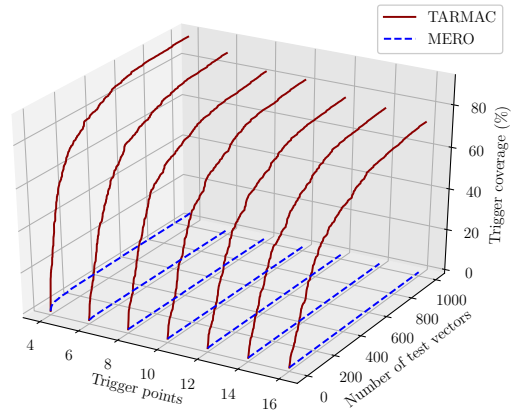
In this experiment, we run Algorithm 4 with 64 threads in parallel to generate the same amount of test vectors as MERO. The results are shown in the last two columns of Table I. Since the overall coverage of TARMAC and TARMAC$_p$ are similar, the coverage of TARMAC$_p$ is omitted.[2] Overall, TARMAC$_p$ with 64 threads can achieve up to 1275x (402x on average) improvement over MERO, and up to 63.8x (50.5x on average) improvement over TARMAC.

To evaluate the utilization of multi-core architectures, we applied Algorithm 4 with different number of threads in MIPS. The result is shown in Figure 9. As we can see, the utilization of multiple cores is very high. Compared to the single thread scenario, 64 threads in parallel can achieve 49.7 times speedup. In other words, the overhead of protecting shared satisfiability graph using mutex in Algorithm 4 is negligible. As we can

[2]The coverage difference of TARMAC and TARMAC$_p$ is from the randomization of sampling. When the random seeds are carefully selected, the coverage can be exactly the same.



(a) c2670 [27]



(b) MIPS Processor [37]

Fig. 8: Trigger condition coverage of TARMAC and MERO on c2670 and MIPS with respect to the number of test vectors given a number of trigger points. The dark red solid line represents TARMAC and the blue dash line represents MERO.

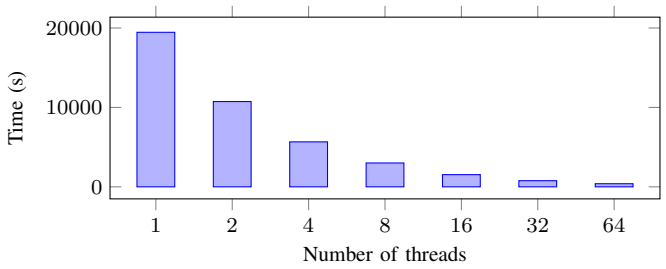


Fig. 9: The test generation time of Algorithm 4 applied in MIPS with different number of threads running in parallel.

see, Algorithm 4 can generate 25042 test vectors for MIPS in approximately 6.5 minutes.

E. Compactness and Efficiency

To compare the compactness and efficiency of TARMAC with MERO, we terminated TARMAC when it just surpassed the same trigger coverage as MERO. In this experiment, we omit the benchmarks s35932 that MERO achieved full coverage, because 100% coverage can be achieved with much fewer test vectors but test length is not a configurable parameter in MERO. It would be an unfair comparison if we compare the test length of TARMAC to the number of s35932 in Table I. The results of the remaining benchmarks are shown in Table II. Note that one test vector in TARMAC can outperform the trigger coverage of MERO for c2670, MEM and MIPS. In all the other benchmarks, the difference of corresponding trigger coverage is minimal.

Table II suggests that test vectors generated by TARMAC are several orders-of-magnitude more compact than MERO. For ISCAS benchmarks, the average reduction of test vectors is in the order of hundreds to achieve the same coverage. The compactness gap becomes larger and larger when the size of design grows. For example, while most of the reductions in small benchmarks (combinational circuits) are within 100 times, the reductions in sequential benchmarks grows to the order of thousands. In MEM and MIPS, the reduction even goes beyond 25 thousands.

The improvement in test generation time follows the same trend as test length reduction. For example, while most of the time improvements in small benchmarks are within the order of hundreds, the improvements in sequential benchmarks grows to the order of thousands even ten thousands. Finally, the improvement in MIPS processor even goes beyond 152 thousands.

From the perspective of debug engineer, efficiency of a test generation approach consists of two aspects. The first one is test generation time. From Table II, we can see that the improvements of test generation time over MERO are several orders of magnitude. The other one is test length as it decides how many simulations or emulations are needed, which dominates debug time. As a result, a compact test set can lead to significant reduction in overall validation effort. Combining both improvements of test generation and reduction of test length as shown in Table II, the efficiency of TARMAC is several orders of magnitude better than MERO.

F. Trigger Coverage

For better illustration of trigger coverage, we ran all benchmarks long enough and plotted the trigger coverage with respect to the number of test vectors in Figure 10. The x-axis represents the number of tests applied to DUTs, and the y-axis represents the percentage of activated trigger conditions. The efficiency in trigger coverage is the gradient of trigger coverage curves. In most of the figures, TARMAC has much steeper slopes than MERO and the curves of random approach are almost flat. The results demonstrated that TARMAC can cover more trigger conditions faster (with significantly less test vectors) than MERO for most of the benchmarks. For example, with 200 test vectors in c2670, TARMAC already activated all the trigger conditions, while MERO only achieved 20% coverage.

These figures reveal that each vector in TARMAC is able to activate more potential trigger conditions than MERO. As stated in Lemma 4, each test vector can cover all the subgraphs of a satisfiable clique. Hence, if one test vector can activate more rare signals, it covers a larger clique and likely to

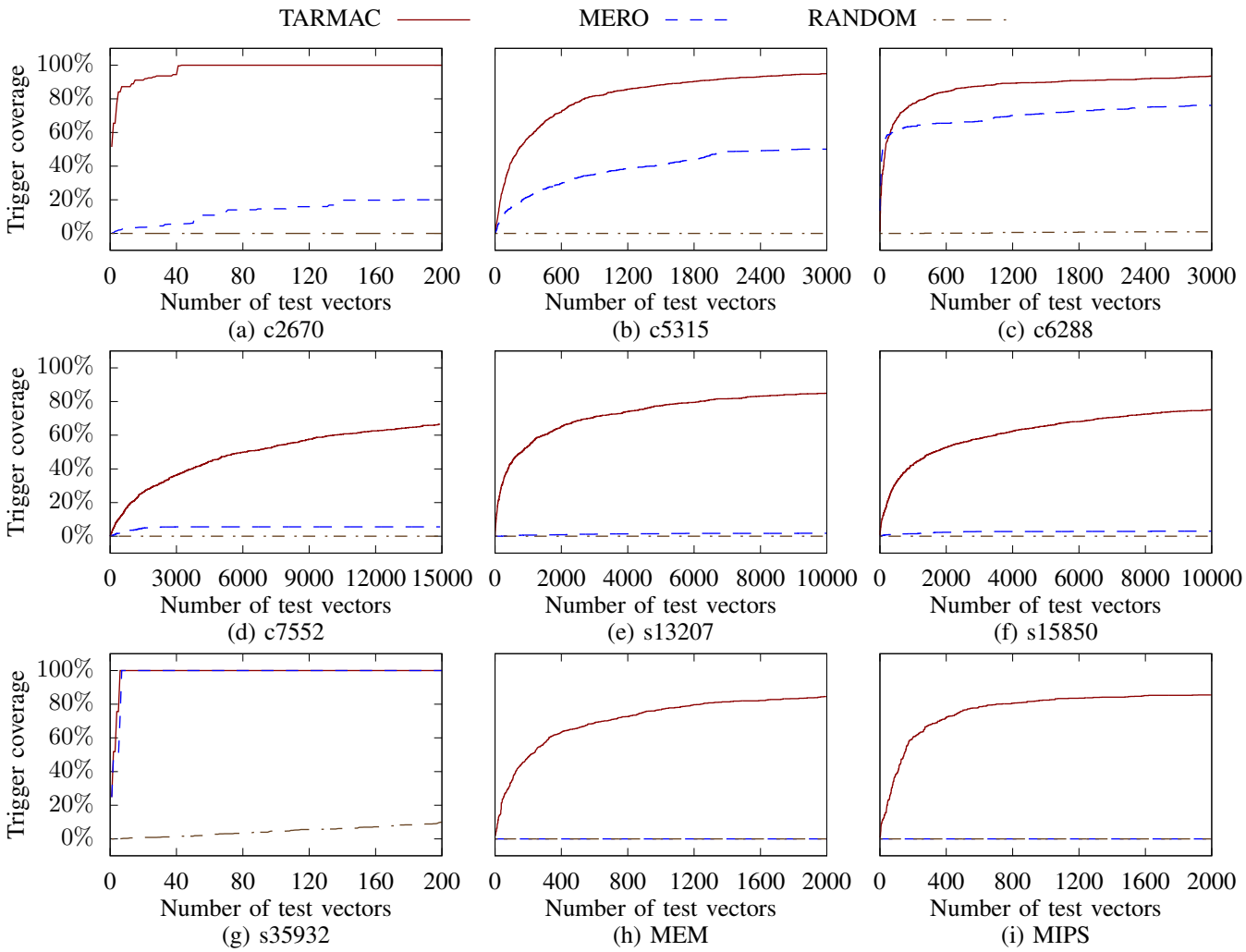


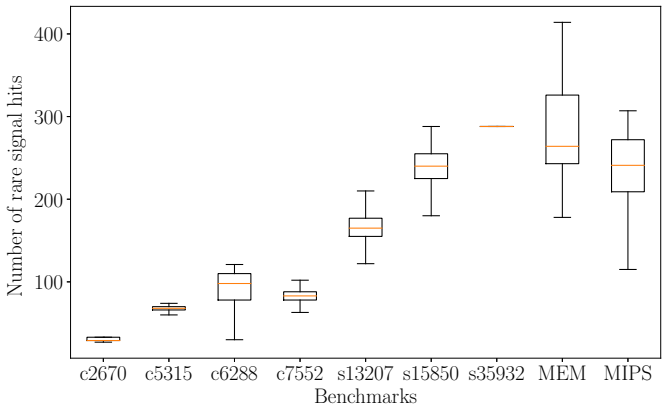
Fig. 10: Trigger coverage with respect to the number of test vectors. TARMAC can achieve full coverage using a small number of test vectors in majority of benchmarks, while MERO and random test vectors can cover only a small fraction of trigger conditions with the same number of test vectors.

activate more potential trigger conditions. Therefore, we define the *quality of a test vector* as the number of rare signals that it can cover (activate). To validate whether the quality of a test vector is the reason for different trigger coverage efficiency, we counted the number of rare signals satisfying their rare values (*rare signal hits*, for short) for each test vector. Figure 11 shows the distribution of rare signal hits by each test vector. The results show that the numbers of rare signal hits are significantly larger in TARMAC (except for the comparable numbers in c6288 and s35932), which is consistent with observations in Figure 10 considering the coverage of trigger conditions. From Algorithm 3, the number of rare signal hits is the same as the size of each sampled maximal satisfiable clique in TARMAC. While in MERO, the number of rare signal hits is the best number of hits after one round of bit flipping from a random test vector. Clearly, the rare signal hits from MERO should be statistically always lower than TARMAC as the rare signal hits in TARMAC are optimal. Moreover, the quality of test vectors in MERO is not guaranteed, since it partially depends on the initial random vectors. As a result, MERO has low rare signal hits (normally less than 50), which is significantly smaller than rare signal hits in TARMAC.

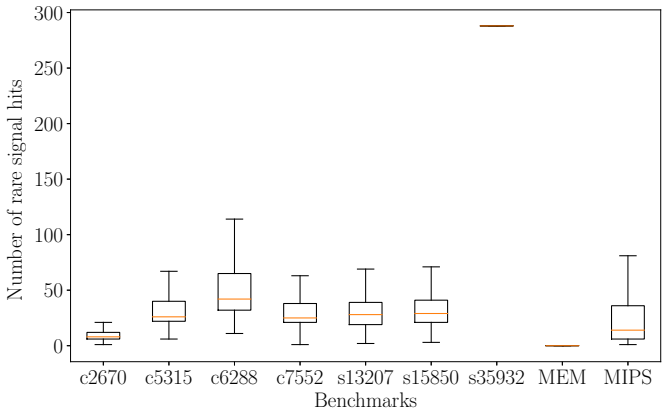
G. Evaluation on TrustHub Benchmarks

To show the performance of the generated tests on detecting hardware Trojans from TrustHub [39], we selected nine gate-level Trojans from three benchmarks (the three largest ISCAS benchmarks from Table I whose Trojan-inserted versions are available in TrustHub), as shown in Table III. The second column shows the performance of the tests generated by only analyzing the golden designs (same as the Section V-C), with no prior knowledge of the design under test (DUTs) with inserted Trojans (referred as "*DUT unknown*"). As we can see, TARMAC can detect all hardware Trojans except one in s13207. It is expected since we are not able to activate all potential hardware Trojans unless we are able to enumerate all satisfiable cliques. Let us consider the following two scenarios based on when a Trojan gets inserted in the design flow.

1) Pre-silicon (DUT-known): In this scenario, we assume that the Trojan gets inserted during the design phase, and therefore, we have access to the Trojan-inserted implementation (referred as "*DUT known*"). To compare with existing



(a) TARMAC (Our Approach)



(b) MERO [3]

Fig. 11: The distribution of rare signal hits by generated test set in all benchmarks.

methods that detect hardware Trojans by analyzing Trojan-infected designs, such as VeriTrust [5], we applied TARMAC on the Trojan-infected designs and evaluated the performance, as shown in the third and fourth columns of Table III. Since only the last three benchmarks are reported in [5], we mark the others with dash lines for VeriTrust. As we can see from the table, our approach is comparable to VeriTrust in the scenarios when the source code of the Trojan-inserted design is known.

2) Post-silicon (DUT-unknown): In this scenario, we assume that the Trojan gets inserted during the manufacturing phase, and therefore, we do not have access to the Trojan-inserted implementation (referred as "*DUT unknown*") during test generation. We applied the same set of tests generated in Section V-C, which are generated by analyzing the golden models, to the nine benchmarks from TrustHub. Our approach detected eight out of the nine benchmarks. The detection rate is very promising considering that we only sampled a small subset of the all possible cliques (future unknown Trojans). Note that VeriTrust [5] is not applicable in this scenario since it requires the availability of a Trojan-inserted netlist for Trojan detection.

Another interesting observation is that even though some trigger points of a hardware Trojan are outside of our PTS, our approach can still activate them. For example, in s15850-T001, there are five signals that are not rare out of 14 trigger points in total. Our approach can activate the Trojan (DUT unknown). The reason is that the probability of the tests (which activate all the rare signals) activating the Trojans is the same as the probability of random tests satisfying all non-rare signals. Therefore, as long as the number of signals outside of PTS is relatively small, our generated tests are likely to activate the Trojans.

TABLE III: Comparison using TrustHub benchmarks

Bench	DUT unknown	DUT known	
	TARMAC	TARMAC	VeriTrust [5]
s13207-T000*	✓	✓	-
s13207-T001*	✓	✓	-
s13207-T002*	×	✓	-
s15850-T000*	✓	✓	-
s15850-T001*	✓	✓	-
s15850-T002*	✓	✓	-
s35932-T100	✓	✓	✓
s35932-T200	✓	✓	✓
s35932-T300	✓	✓	✓

*These benchmarks are from TRIT-TC [40] in TrustHub.

VI. Conclusion

Trigger activation is a fundamental challenge in detection of hardware Trojans. While prior efforts using statistical test generation are promising, they are neither scalable for large designs nor suitable for activating extremely rare trigger conditions in stealthy Trojans. In this paper, we introduced a new paradigm to solve trigger activation problem. This paper made the following important contributions. 1) Our approach is the first attempt in mapping the problem of test generation for trigger activation to the problem of covering maximal satisfiability cliques. 2) We proved that valid trigger conditions and satisfiability cliques are one-to-one mapping. We also proved that the test vectors generated by our paradigm are both complete and compact. 3) We presented efficient test generation algorithms to repeatedly sample maximal satisfiability cliques and generate a test vector for each of them. We explored the effectiveness of random sampling, lazy construction as well as multi-threading to improve the test generation efficiency. Our experimental results demonstrated that our approach is both scalable and effective in generating efficient test vectors for a wide variety of trigger conditions. Our approach outperforms the state-of-the-art techniques by several orders-of-magnitude in terms of trigger coverage, test length as well as test generation time. Our test generation algorithms can be utilized for activating extremely rare trigger conditions to fulfill diverse requirements such as improvement of functional (trigger) coverage as well as side-channel sensitivity.

Acknowledgments

This work was partially supported by the National Science Foundation (NSF) grant CCF-1908131.

References

[1] A. Mosenia and N. K. Jha, "A comprehensive study of security of internet-of-things," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 4, pp. 586–602, Oct 2017.

[2] Y. Lyu and P. Mishra, "Automated trigger activation by repeated maximal clique sampling," in *Asia and South Pacific Design Automation Conference (ASPDAC), Beijing, China, January 13 - 16*, 2020.

[3] R. S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, and S. Bhunia, "Mero: A statistical approach for hardware trojan detection," in *Cryptographic Hardware and Embedded Systems*, 2009, pp. 396–410.

[4] A. Waksman, M. Suozzo, and S. Sethumadhavan, "Fanci: Identification of stealthy malicious logic using boolean functional analysis," in *ACM Conference on Computer Communications Security*, 2013, pp. 697–708.

[5] J. Zhang, F. Yuan, L. Wei, Y. Liu, and Q. Xu, "Veritrust: Verification for hardware trust," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 7, pp. 1148–1161, 2015.

[6] A. Bazzazi, M. T. Manzuri Shalmani, and A. M. Hemmatyar, "Hardware trojan detection based on logical testing," *J. Electron. Test.*, vol. 33, no. 4, pp. 381–395, Aug. 2017.

[7] M. A. Nourian, M. Fazeli, and D. Hely, "Hardware trojan detection using an advised genetic algorithm based logic testing," *Journal of Electronic Testing*, vol. 34, no. 4, pp. 461–470, Aug 2018.

[8] Y. Lyu and P. Mishra, "Automated test generation for trojan detection using delay-based side channel analysis," in *Design Automation and Test in Europe (DATE), Grenoble, France, March 9 - 13*, 2020.

[9] ——, "A survey of side-channel attacks on caches and countermeasures," *Journal of Hardware and Systems Security*, vol. 2, no. 1, pp. 33–50, Mar 2018.

[10] ——, "Efficient test generation for trojan detection using side channel analysis," in *Design Automation and Test in Europe (DATE), Florence, Italy, March 25 - 29*, 2019.

[11] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan detection using ic fingerprinting," in *2007 IEEE Symposium on Security and Privacy (SP '07)*, May 2007, pp. 296–310.

[12] Y. Jin and Y. Makris, "Hardware trojan detection using path delay fingerprint," in *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, June 2008, pp. 51–57.

[13] G. Zarrinchian and M. S. Zamani, "Latch-based structure: A high resolution and self-reference technique for hardware trojan detection," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 100–113, Jan 2017.

[14] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Design Automation Conference*, 2003, pp. 338–342.

[15] D. Rai and J. Lach, "Performance of delay-based trojan detection techniques under parameter variations," in *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*, 2009, pp. 58–65.

[16] J. He, X. G. Y. Zhao, and Y. Jin, "Hardware trojan detection through chip-free electromagnetic side-channel statistical analysis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2939–2948, October 2017.

[17] I. Pomeranz and S. M. Reddy, "A measure of quality for n-detection test sets," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1497–1503, Nov 2004.

[18] M. E. Amyeen, S. Venkataraman, A. Ojha, and S. Lee, "Evaluation of the quality of n-detect scan atpg patterns on a processor," in *2004 International Conference on Test*, Oct 2004, pp. 669–678.

[19] P. Kitsos, D. E. Simos, J. Torres-Jimenez, and A. G. Voyiatzis, "Exciting fpga cryptographic trojans using combinatorial testing," in *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, 2015, pp. 69–76.

[20] Y. Lyu and P. Mishra, "Automated test generation for activation of assertions in rtl models," in *Asia and South Pacific Design Automation Conference (ASPDAC), Beijing, China, January 13 - 16*, 2020.

[21] Y. Lyu, A. Ahmed, and P. Mishra, "Automated activation of multiple targets in rtl models using concolic testing," in *Design Automation and Test in Europe (DATE), Florence, Italy, March 25 - 29*, 2019.

[22] Y. Lyu, X. Qin, M. Chen, and P. Mishra, "Directed test generation for validation of cache coherence protocols," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 1, pp. 163–176, 2019.

[23] D. A. Mathaikutty, , and and, "Model-driven test generation for system level validation," in *2007 IEEE International High Level Design Validation and Test Workshop*, Nov 2007, pp. 83–90.

[24] F. Wolff, C. Papachristou, S. Bhunia, and R. S. Chakraborty, "Towards trojan-free trusted ics: Problem analysis and detection scheme," in *2008 Design, Automation and Test in Europe*, March 2008, pp. 1362–1365.

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

[26] H. D. Foster, "Trends in functional verification: A 2014 industry study," in *Proceedings of the 52Nd Annual Design Automation Conference*, ser. DAC '15, 2015, pp. 48:1–48:6.

[27] "ISCAS85 combinational benchmark circuits," https://filebox.ece.vt.edu/~mhsiao/iscas85.html.

[28] A. Ahmed, F. Farahmandi, Y. Iskander, and P. Mishra, "Scalable hardware trojan activation by interleaving concrete simulation and symbolic execution," in *IEEE International Test Conference (ITC)*, 2018.

[29] Y. Lyu and P. Mishra, "Scalable concolic testing of rtl models," *IEEE Transactions on Computers*, 2020.

[30] "ISCAS89 sequential benchmark circuits," https://filebox.ece.vt.edu/~mhsiao/iscas89.html.

[31] R. M. Karp, *Reducibility among Combinatorial Problems*, 1972, pp. 85–103.

[32] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo, *The Maximum Clique Problem*. Boston, MA: Springer US, 1999, pp. 1–74.

[33] J. W. Moon and L. Moser, "On cliques in graphs," *Israel Journal of Mathematics*, vol. 3, no. 1, pp. 23–28, Mar 1965.

[34] C. Bron and J. Kerbosch, "Algorithm 457: Finding all cliques of an undirected graph," *Commun. ACM*, vol. 16, no. 9, pp. 575–577, Sep. 1973.

[35] M. C. Schmidt, N. F. Samatova, K. Thomas, and B.-H. Park, "A scalable, parallel algorithm for maximal clique enumeration," *Journal of Parallel and Distributed Computing*, vol. 69, no. 4, pp. 417 – 428, 2009.

[36] Y. Huang, S. Bhunia, and P. Mishra, "Scalable test generation for trojan detection using side channel analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 11, pp. 2746–2760, Nov 2018.

[37] "Opencores," https://www.opencores.org/.

[38] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2008, pp. 337–340.

[39] "Trusthub," https://www.trust-hub.org/, accessed: 2018-10-10.

[40] J. Cruz, Y. Huang, P. Mishra, and S. Bhunia, "An automated configurable trojan insertion framework for dynamic trust benchmarks," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 1598–1603.

Yangdi Lyu received his B.E. degree from Department of Hydraulic Engineering, Tsinghua University, Beijing, China in 2011, and his Ph.D. degree from the Department of Computer and Information Sciences and Engineering, University of Florida in 2020. His research interests include the development of test generation techniques for hardware trust, the security validation of system-on-chip, and microarchitectural side-channel analysis.

Prabhat Mishra (SM'08) is a Professor in the Department of Computer and Information Science and Engineering at the University of Florida. His research interests include embedded and cyber-physical systems, hardware security and trust, energy-aware computing, formal verification, and system-on-chip security validation. He received his Ph.D. in Computer Science and Engineering from the University of California, Irvine. He has published 8 books, 25 book chapters, and more than 150 research articles in premier international journals and conferences. His research has been recognized by several awards including the NSF CAREER Award, IBM Faculty Award, three best paper awards, and EDAA Outstanding Dissertation Award. Prof. Mishra currently serves as an Associate Editor of ACM Transactions on Design Automation of Electronic Systems and IEEE Transactions on VLSI Systems. He is an ACM Distinguished Member and a Senior Member of IEEE.