Scalable Detection of Hardware Trojans using ATPG-based Activation of Rare Events

Aruna Jayasena, *Student Member, IEEE,* and Prabhat Mishra, *Fellow, IEEE,*

***Abstract*—Semiconductor supply chain vulnerability is a major concern in designing trustworthy systems. Malicious implants, popularly known as hardware Trojans, can get introduced at different stages in the System-on-Chip (SoC) design cycle. While there are promising test generation techniques for hardware Trojan detection, they have two practical limitations: (i) these approaches are designed to activate rare states while ignoring rare transitions, and (ii) these approaches are not scalable for large designs. In this paper, we propose a scalable test generation framework to address the above challenges. Our threat model assumes that an adversary may exploit rare events consisting of rare signals (states) as well as rare branches (transitions). We show that the rare branch coverage problem can be mapped to the rare signal coverage problem. We propose a scalable framework for detecting hardware Trojans using Automated Test Pattern Generation (ATPG) based activation of rare events. Specifically, we utilize the complementary abilities of N-detection and maximal clique activation of rare events to generate efficient test patterns. Experimental evaluation shows that our ATPG-based framework is scalable and significantly outperforms the state-of-the-art test generation based Trojan detection techniques.**

***Index Terms*—Hardware security, Trojan Detection, ATPG**

I. Introduction

Semiconductor companies rely on the global supply chain for the development of System-on-Chip (SoC) designs. This involves increasing utilization of third-party Intellectual Property (IP) cores as well as outsourcing of various design automation activities including validation, synthesis, layout, and fabrication. An attacker has various opportunities to introduce hardware Trojans (HT) by exploiting the supply chain vulnerabilities.

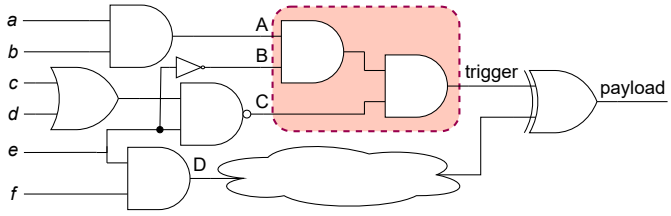


Fig. 1: Hardware Trojan triggered by three rare signals (A,B,C)

A. Threat Model

Our threat model assumes that an adversary can introduce hardware Trojans (HT) during any stages of the SoC development cycle including the design of RTL models, synthesis to gate-level netlist, and fabrication. An HT consists of two major components: trigger and payload. An adversary is likely to construct the trigger such that it will avoid detection during traditional validation using millions of random or constrained-random test patterns. A stealthy trigger can be constructed using rare events such as rare signals (states) or rare branches (transitions). Figure 1 shows an example Trojan circuit with a trigger and payload. When the trigger gets activated, the payload can enable malicious activities such as information leakage, incorrect execution, or denial-of-service. In the example circuit, the Trojan payload flips the expected output.

A. Jayasena and P. Mishra are with the Department of Computer & Information Science & Engineering, University of Florida, Gainesville, Florida.

B. Limitations of Existing Methods

There are a wide variety of HT detection methods. Many recent approaches rely on generating test patterns using golden models (e.g., TLM or RTL models) and applying them on gate-level implementation (assumes HT introduction during synthesis) or integrated circuits (assumes HT introduction during fabrication). Figure 2 shows a broad overview of these methods. The recent test generation methods can be mainly divided into two categories: (i) statistical test generation relies on N-detect principle [1] that tries to activate each rare signal N times [2], and (ii) maximal clique sampling tries to activate as many rare signals as possible using a single test [3].

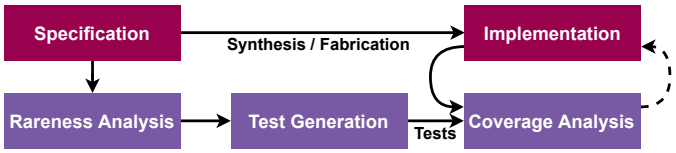


Fig. 2: Overview of test generation based Trojan detection

While the existing approaches are promising in increasing the likelihood of activating the unknown and stealthy trigger, they have two major limitations: scope and scalability. These approaches have a limited scope since they focus on rare signals (states), but an attacker may exploit both rare signals (states) as well as rare branches (transitions). Moreover, these approaches are not scalable for large designs since the underlying bit-flipping (statistical) or constraint solving (computing expression for cliques) algorithms are exponential with respect to the design (and input) complexity. In fact, statistical is likely to violate the N-detect principle. As shown in Figure 3, some rare nodes are activated too many times while the rarest ones are not activated at all by MERO [2], while our proposed ATPG-based approach (ND-ATPG) activates each rare node N times. On the other hand, the time complexity of the state-of-the-art maximal clique activation technique (TARMAC [3]) grows significantly with the increase in the number of rare nodes. Figure 4 shows the satisfiability graph

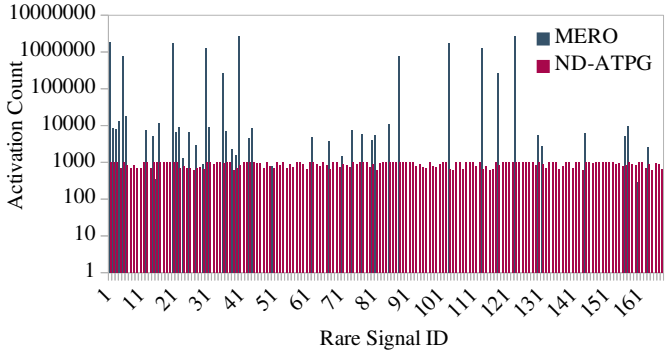


Fig. 3: Rare node activation count of MERO [2] compared with ND-ATPG for N=1000 on elliptic curve cryptography module. It shows the number of times each signal got activated.

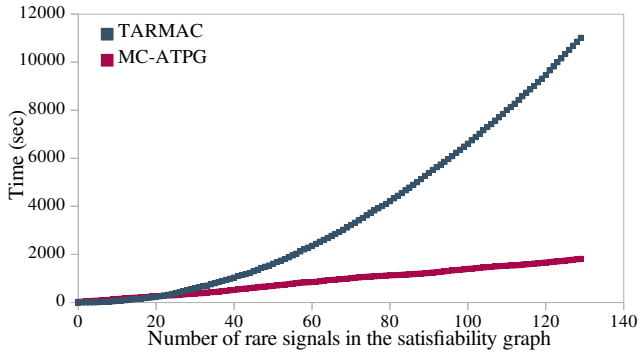


Fig. 4: SATisfiability graph generation time of TARMAC [3] (with lazy construction) compared with MC-ATPG for *eeageneric* module in elliptic curve cryptography.

generation time of TARMAC compared with our proposed ATPG-based maximal clique activation (MC-ATPG). It can be observed that MC-ATPG significantly outperforms TARMAC in terms of scalability.

C. Research Contributions

In this paper, we propose an efficient test generation based HT detection framework that addresses the above challenges. Our threat model assumes that an adversary may exploit rare events consisting of rare signals (states) as well as rare branches (transitions). We utilize Automated Test Pattern Generation (ATPG) for N-activation as well as maximal clique activation of rare events. Our research needs to answer five important questions: (i) how to utilize ATPG to activate rare branches, (ii) what is the rationale for N-activation of rare signals and branches, (iii) how to use ATPG for activating a rare event (rare signal or rare branch) multiple times since ATPG provides only one test for a given fault, (iv) how to construct satisfiability graph with ATPG, and (v) how to activate cliques of rare events with ATPG. To address the first challenge, we map the branch coverage problem to stuck-at coverage problem (Section IV-B). To answer the second question, we have to realize that the output of the trigger may be the rarest signal (or branch), but it may have been introduced during fabrication. In other words, if the trigger was in the RTL design, activating each rare signal (or branch) only once would have activated the trigger, and the designer should have removed it during the design phase. To address the third question, we automatically generate constraints based on the previously generated test so that it can provide a new test in the next iteration for the same stuck-at fault (Section VI-A). The fourth and fifth challenges are addressed by implementing stuck-at faults combined with output port constraints by connecting all rare events to the outputs of the design (only for test generation purposes) (Section VI and Section VII). Specifically, this paper makes the following major contributions.

- We show that the rare branch coverage problem can be mapped to the rare signal coverage problem.
- To the best of our knowledge, there are no prior test generation efforts for N-activation of rare branches.
- We propose a scalable framework for N-activation of rare events (i.e., rare signals and rare branches) using ATPG.
- We propose a complementary and scalable framework for activating maximal cliques of rare events using ATPG.
- Experimental results show that our ATPG-based framework is scalable and significantly outperforms the state-of-the-art test generation based HT detection techniques.

The remainder of this paper is organized as follows. Section II surveys the related approaches to highlight the novelty of our work. Section III provides an overview of the proposed methodology. Section IV presents the steps involved in rareness analysis. Section V outlines the fault modeling for rare events followed by our proposed ATPG-based HT detection framework using N-detection of rare events (Section VI) as well as maximal clique activation (Section VII). Section VIII describes the framework to generate HT-injected benchmarks and utilizing them to evaluate the quality of the generated test patterns. Section IX presents the experimental results. Finally, Section XI concludes the paper.

II. Background and Related Work

Random and constrained random tests are widely used during traditional functional validation methodology. Due to the exponential input space complexity of the designs, even billions or trillions of test vectors are not enough to validate all possible functional scenarios in today's industrial designs. Directed tests are promising to cover the remaining scenarios as well as the corner cases that are not activated by random and constrained-random tests. Manual development of directed tests is time-consuming, error-prone, and can be infeasible for complex designs. There are effective approaches for automated generation of directed tests [4]–[8] that can be utilized for activating targeted scenarios. While these approaches are useful for validation of functional scenarios, they are not suitable for detecting security vulnerabilities such as hardware Trojans. This is due to the fact that stealthy Trojans consists of extremely rare trigger conditions.

There are a wide variety of approaches for the detection of hardware Trojans (HT) that can be divided into three broad categories: test generation (simulation-based validation) [2], [3], [9], side-channel analysis [10]–[15], and machine learning (ML) [16]. ML-based approaches can be further subdivided based on whether they require golden model (supervised learning) or not (unsupervised learning). In this paper, we focus on test generation based HT detection. This assumes

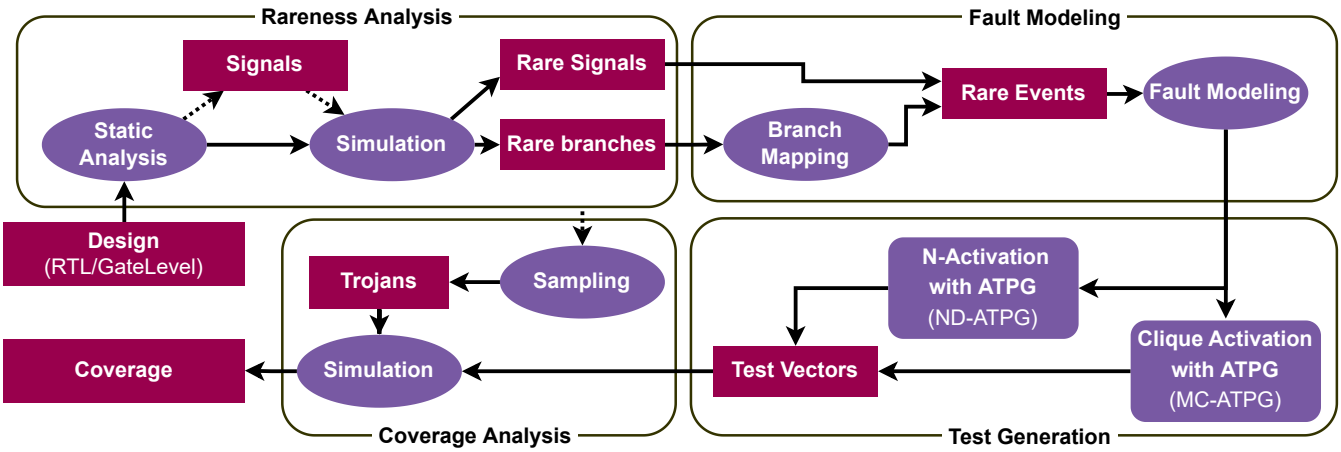


Fig. 5: Overview of our proposed ATPG-based activation of rare events for hardware Trojan detection. It consists of four major steps: rareness analysis, fault modeling, test generation, and coverage analysis. We propose two test generation methods: N-activation of rare events using ATPG (ND-ATPG) and ATPG-based activation of maximal cliques (MC-ATPG).

that the activation of HT will lead to a functional mismatch. In other words, if the generated test can activate the HT trigger during simulation (execution), we will be able to detect the HT by comparing the output with the expected output. There are two complementary avenues for test generation based HT detection: statistical test generation [2] and maximal clique sampling [3].

Statistical test generation approaches rely on N-detect principle [1] that tries to activate each rare node N times. Assuming an adversary is likely to construct a trigger consisting of rare nodes, the goal of the statistical approaches is to maximize the likelihood of activating the unknown trigger. Unfortunately, the statistical approach is not scalable for large designs due to the exponential nature of its bit-flipping based algorithm (MERO [2]) as demonstrated in Figure 11. Most importantly, it typically violates N-detect policy since it cannot activate all the rare nodes in a reasonable time. For example, Figure 3 shows that MERO can never activate some of the rare nodes even for simple designs. While Pan et al. [9] improved the limitations of the bit-flipping algorithm using reinforcement learning (TGRL), it still faces the scalability problem. There are various research efforts that utilizes MERO [2] as the initial test pattern generator for side-channel based test generation [11]–[14]. The N-detect concept has been used for isolation of suspected signal for equivalence checking [17]. Physically aware test selection using N-detect principle is utilized to improve the defect coverage in manufacturing testing [18], [19]. *None of these approaches are used for activating triggers with rare signals as well as rare branches in complex pre-silicon designs.*

Maximal clique sampling (TARMAC [3]) attacks the problem from the opposite direction. Unlike statistical approaches that try to activate each node N times, TARMAC tries to activate as many rare nodes as possible using a single test by covering the maximal cliques in a graph consisting of rare nodes. Specifically, it constructs the logical expressions of each rare node and checks the satisfiability of multiple expressions corresponding to a clique in the graph. *This approach is not scalable since the complexity of the expression generation as well as satisfiability solving grow exponentially with the design size.* For example, Figure 4 shows the exponential nature of graph generation time with TARMAC.

There are promising efforts in utilizing ATPG for HT detection [20] that has explored a hybrid solution consisting of ATPG and model checking for HT detection in full-scan as well as partial-scan designs. *To the best of our knowledge, our proposed effort is the first attempt at utilizing ATPG for activation of rare events (signals and branches) for efficient and scalable detection of hardware Trojans.*

III. ATPG-based Activation of Rare Events

Figure 5 provides a high-level overview of our proposed methodology. First, it performs rareness analysis by simulating with millions of random tests. Specifically, it marks the signals and branches that are activated less than a specific threshold. Next, it maps the rare branch coverage problem to rare signal coverage problem to generate rare events. These rare events are converted to stuck-at faults to enable ATPG-based test generation. Specifically, we propose two complementary ATPG-based test generation approaches: (i) ATPG-based N-activation of rare events (ND-ATPG), and (ii) ATPG-based maximal clique activation (MC-ATPG) to generate test vectors. Finally, it evaluates the quality of the generated tests by computing the coverage of the detected Trojans using Trojan-embedded benchmarks.

Algorithm 1 Activation of Rare Events with ATPG

Input design D, rarenessThreshold r, N
Output testVectors T

1: $\{R_s, R_b\} \leftarrow rarenessAnalysis(D, r)$
2: $\{D', R_{b2s}\} \leftarrow mapBranch2Signal(D, R_b)$
3: $S_f \leftarrow faultModeling(R_s, R_{b2s})$
4: $Tests \leftarrow testGeneration(D', S_f, N)$
5: $coverageAnalysis(D, Tests)$
6: Return $Tests$

Algorithm 1 highlights the four major steps in our framework: rareness analysis, fault modeling, test generation for activation of rare events, and coverage analysis. The remainder of this section describes these steps in detail.

IV. Rareness Analysis

As discussed in Section I-A, attackers are likely to use extremely rare (hard-to-detect) behaviors to trigger the attack. If an attacker constructs a trigger using non-rare (easy-to-activate) behaviors, the trigger is likely to be activated during traditional functional validation, and therefore, the Trojan will be easily detected. With rareness analysis, we try to highlight the corner cases that are not validated by functional validation. We take the design (RTL or the gate-level models) and simulate for an adequate number of random simulations. All the hierarchical designs are flattened before the simulation to ensure that each signal has a unique identity. For synchronous designs, the number of clock cycles to simulate the design is determined by the design pipeline depth. For example, in case of AES benchmark, the pipeline depth is 21 clock cycles. Therefore, it is important to do each simulation at least 21 cycles for the rareness analysis to get a set of useful rare events in case of AES benchmark. The remainder of this section describes three important tasks in rareness analysis: rareness analysis of signals, elimination of false rare signals, and rareness analysis of branches.

A. Rareness Analysis for Signals

During simulation, we monitor every signal for its values. For multi-bit variables in RTL designs, we monitor each bit individually. Next, we calculate the rareness value for each variable for the occurrence of '1' and '0'. We use a rareness threshold to filter the rarest signals from all the signals. Consider the simplified code snippet shown in Listing 1 as an illustrative example. Assume '0.2' as the rareness threshold. Here assignment to the register 'a' is under a rare branch. The probability of 'key' having the value of 255 as shown in line 8 is 0.004 which makes it a rare event. Assignment to the wire '$N13$' happens based on the output of the rare branch which makes '$N13$' having the value '0' even rarer. During the simulation, we monitor the value of '$N13$' to calculate its rareness for different values. For example, we can calculate the rareness of $\{N13, 0\}$ by dividing the number of instances '$N13$' appeared as '0' by the total number of appearances of '$N13$'. According to the example, in Listing 1, occurrence of $N13$ appearing as '0' is below the rareness threshold which makes the signal $\{N13, 0\}$ a rare event.

1) Elimination of False Positive Rare Signals: The goal of this step is to remove signals that may appear as rare during simulation but they are not rare (e.g., they have fixed values). Consider AES benchmark that contains a multi-bit signal named as *rcon* which is assigned to a fix constant *hex* value. During simulation, similar signals do not change their pre-assigned constant values unless they are updated later by a different assignment. Since they are non-rare signals, an attacker is unlikely to use them to construct a trigger. Therefore, eliminating such signals will lead to improved rareness analysis. Note that we cannot simply remove signals with rareness value of 0 since rareness of 0 can also be caused by insufficient number of simulation iterations. In order to eliminate such false positive signals, we perform static data-flow analysis. First, we construct the Data Flow Graph (DFG) of the design. Next, we eliminate all static signals. Finally, we recursively go through the DFG to identify and remove signals from the set of potential rare nodes where the predecessor signal is a constant.

Listing 1: Illustrative example for rare events

```
1  module simpleAes (clk, rst, key);
2      input       clk, rst;
3      input       [7:0] key;
4      output reg [7:0] state_out;
5      reg [1:0]  a, b, c, d;
6      wire N13;
7      always @ (posedge clk) begin
8          if  (key=8'd255)
9              a <= a + 1'b1;
10     end
11     assign N13 = (a[1])? 1'b0 : 1'b1;
12 endmodule
```

Listing 2: Branch tagging example

```
1  module simpleAes (clk, rst, key);
2      ...
3      reg t0;
4      always @ (posedge clk) begin
5          if (rst)
6              t0<=1'b0;
7          if  (key=8'd255)
8              a <= a + 1'b1;
9              t0<=1'b1;
10     end
11     ...
12 endmodule
```

B. Rareness Analysis for Branches

This case is applicable for designs with branches (e.g., RTL models). During each random simulation, we monitor the status of each branch whether it was taken or not. The branches that were activated less than the rareness threshold during the random simulation are considered as rare branches. In the example of Listing 1, the probability of branch in line 8 activating under random simulation is low and hence it will be considered as a rare branch (rare event).

V. Fault Modeling for Rare Events

Both rare signals and rare branches (discussed in Section IV) are rare events. In this section, we discuss how these rare events are mapped to stuck-at fault before activating them using ATPG to generate the test patterns. We first map the rare branch coverage problem to rare signal coverage problem (Section V-A). Next, we describe fault modeling of rare events (Section V-B).

A. Automated Mapping of Rare Branches to Rare Signals

Once rare branches are identified in Section IV-B, we use a unique tag (identifier signal) for each rare branch. Algorithm 2 presents the methodology of implementing the tag signal – if the branch is taken, the tag value is set to '1', otherwise the tag value is set to '0'. A rare branch with tagging is shown

Listing 3: Instrumented design before synthesis for the illustrative example provided in Listing 1

```
13  module simpleAes (clk, rst, key, t0, N13, a_1);
14      input       clk, rst;
15      input       [7:0] key;
16      output reg [7:0] state_out;
17
18      output t0, N13, a_1;
19      assign a_1 = a[1];
20      reg t0;
21      reg [1:0]  a,  b,  c,  d;
22      wire N13;
23      always @ (posedge clk) begin
24          if (rst)
25              t0<=1'b0;
26          if  (key=8'd255)
27              a <= a + 1'b1;
28              t0<=1'b1;
29      end
30      assign N13 = (a[1])? 1'b0 : 1'b1;
31  endmodule
```

in Listing 2. Here '$t0$' is the tag signal and it is assigned to value '1' under the rare branch in line 9.

Algorithm 2 mapBranch2Signal

Input design D, branch R_b
Output tag R_{b2s}, design D'

1: $R_{b2s} \leftarrow \emptyset$
2: **for** Each $b_i \in R_b$ **do**
3: $D' \leftarrow D.append(t_i)$ ▷ add signal t_i to the design
4: $R_{b2s}.append(t_i)$
5: **for** lines in D **do**
6: **if** $reset$ block in line **then** ▷ Locate reset block
7: $D'.append(t_i \leftarrow 1'b0)$ ▷ Initialize t_i to 0
8: **end if**
9: **if** b_i in line **then** ▷ locate the branch b_i
10: $D'.append(t_i \leftarrow 1'b1)$ ▷ Set t_i to 1
11: **end if**
12: **end for**
13: **end for**
14: Return D', R_{b2s}

B. Stuck-at Fault Modeling for Rare Events

The goal of this section is to generate stuck-at faults for the rare events consisting of rare signals (derived from Section IV-A) as well as rare branches (associated rare tags derived from Section V-A). For all the rare events, corresponding nets are connected to the design output so that the net names are preserved after the synthesis. Instrumented design for the illustrative example is provided in Listing 3. The design is synthesized to a gate-level netlist to prepare it for ATPG-based test generation. We also need to generate stuck-at faults for each of the rare events. Table I shows two example rare nodes ($N13$ from Listing 1 and $t0$ from Listing 2) converted to stuck-at-fault model.

VI. ND-ATPG: Test Generation using ATPG-based N-Activation of Rare Events

Once all the rare events are converted to the stuck-at fault model, we need to obtain N test vectors for each stuck-at fault to satisfy N-detect principle [1] such that each rare event is activated at least N times (each test activates them once). However, ATPG tool produces one test vector for one fault. In this section, we explore how we can obtain N test vectors for each fault using ATPG.

TABLE I: Example rare events with relevant stuck-at-fault expression for generation of test vectors using ATPG

Rare Node		Stuck-at-fault
Net	Value	
t0	1	add_faults -stuck 0 t0
N13	0	add_faults -stuck 1 N13

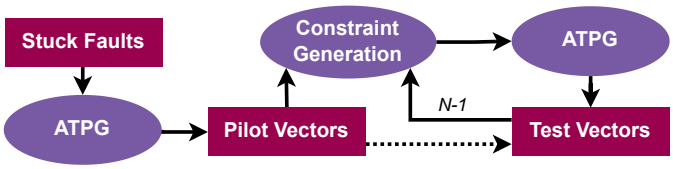


Fig. 6: Overview of N-activation of rare events using ATPG

A. N-Activation of Rare Events with ATPG

Test generation for N-activation is carried out in two steps. The first step generates the initial test vectors (one test for each fault) which are referred as pilot vectors. The second step generates the remaining N-1 vectors (for each fault) based on the pilot vectors with constraints as shown in Figure 6.

Algorithm 3 testGeneration with ND-ATPG

Input design D, faultList Sf, N
Output testVectors T

1: $T \leftarrow \emptyset$
2: **for** Each $f \in Sf$ **do**
3: $T[f] \leftarrow ATPG(D_{(G)}, f)$ ▷ Generate pilot vectors
4: **end for**
5: **for** Each $f \in Sf$ **do**
6: $t_f \leftarrow T[f]$
7: $C_f \leftarrow \emptyset$
8: **for** $i = 0 \rightarrow N-1$ **do**
9: $c_i \leftarrow getConstraints(t_f, D.p_i, C_f)$
10: $t_f \leftarrow ATPG(D, c_i, f)$
11: $C_f.append(c_i)$
12: $T[f][i] \leftarrow t_f$ ▷ Second generation test vectors
13: **end for**
14: **end for**
15: Return T

Algorithm 3 outlines the major steps involved in generation of test vectors. For each stuck-at-fault, we invoke ATPG to generate the pilot vector corresponding to the specific fault (lines 2-4). In order to generate the remaining N-1 vectors per fault, we generate constraints from the previous test to generate the next test (lines 8-13). Algorithm 4 outlines the major steps for generating the constraints. With constraint generation, we limit the input pattern for certain input ports of the design by referring to the previously generated test. $N-1$ constraints are generated such that every constraint will generate a new test vector to activate the same stuck-at-fault. Therefore, we can achieve N-activation of the same rare event with different test

vectors. For the illustrative example (Listing 1), assume that the sample pilot vector for the stuck-at-fault '$\{-stuck\ 0\ t0\}$" is '11111111'. This vector is corresponding to the input port '$key[7\ :\ 0]$'. In order to generate the remaining (second-generation) test vectors, one possible constraint is to flip the $key[7]$ (MSB) and provide it as a constraint to the ATPG tool. The sample constraint for the above example would be "*add_pi_constraints 0 key[7]*". Algorithm 4 will always provide a new constraint until N-1 tests are generated (in addition to the pilot vector).

Algorithm 4 getConstraints

Input testVector t_x, inputPorts p_i, constraintList C_f
Output constraint c_x

for t in t_x **do**
  $c_x \leftarrow \{p_i^t = t^{-1}\}$ ▷ Generate a new c_x for f
  if $c_x \in C_f$ **then**
    $c_x \leftarrow getConstraints(t_f, p_i, C_f)$
  else
    Return c_x
  end if
end for

VII. MC-ATPG: ATPG-based Maximal Clique Activation

The previous section discussed N-activation methodology where we try to activate each rare event N times hoping that N test vectors will trigger different Trojans that consist of a specific combination of rare events chosen by the attacker. In this section, we propose a complementary approach where we try to activate all the trigger combinations at once. This method will produce more effective and compact test vectors with the cost of time and effort. Figure 7 provides an overview of the proposed test generation framework using ATPG-based maximal clique activation.

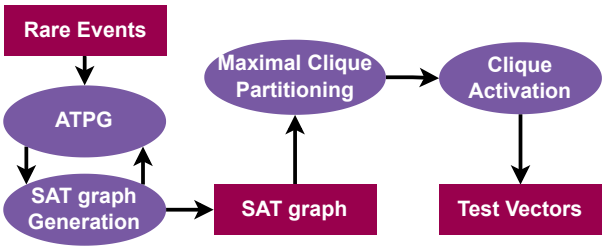


Fig. 7: Overview of maximal clique activation of rare events

Algorithm 5 outlines the the major steps in Figure 7: satisfiability graph generation, maximal clique partitioning, and test generation using ATPG-based maximal clique activation. The remainder of this section describes these steps in detail.

A. Satisfiability Graph Generation

The construction of the satisfiability graph utilizes the output from the rareness analysis (section IV) as well as the instrumented design from the fault modeling (Section V-B). Each rare event (node) is a vertex of the satisfiability graph.

Algorithm 5 testGeneration with MC-ATPG

Input instrumentedDesign D, faultList Sf
Output testVectors T

$G \leftarrow getSATisfiabilityGraph(D, Sf)$
$cliques \leftarrow maxCliquePartition(g)$
$sort(cliques)$
for c in $cliques$ **do**
  $T \leftarrow T \cup activateClique(D, c)$
end for
Return T

An edge between those two rare nodes in the satisfiability graph implies that the respective logical expressions can be satisfied. For example, Figure 8 shows the satisfiability graph constructed from the circuit in Figure 1. In the extreme case (all the rare nodes are satisfiable), we will have a complete graph with $n \times \frac{n-1}{2}$ edges, where n is the number of nodes in the graph. For example, Figure 8 has an edge between A and B since the logic expressions associated with them can be satisfied by feeding the input pattern '110100' to the circuit in Figure 1. There is no edge between B and D because the input e has conflicting requirements (D expects it to be '1' while B expects it to be '0') which cannot be satisfied simultaneously.

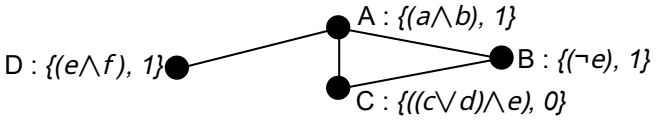


Fig. 8: Satisfiability graph with four nodes of the circuit in Fig. 1 (with logic expressions and rare values in parentheses)

Due to the internal structure of the circuit, the nature of the graph can vary. For example, in case of ISCAS2670 benchmark, the satisfiability graph is *dense* (Figure 9a) since the rare signals closely related and well connected. On the other hand, the rare events are scattered across the AES 128 benchmark that leads to a *sparse* graph as shown in Figure 9b. If the rare signals are located together as a single cluster, the output is a *dense* graph. Usually in large designs, we can find dense clusters scattered in the graph. For improved accuracy it is important to construct the satisfiability graph by querying all possible two trigger combinations ($n \times \frac{n-1}{2}$ of queries).

In order to reduce the overall time in handling exponential number of queries, we read the design with the ATPG tool and perform specific pre-processing steps. Later all the queries go through this model file and redundant pre-processing steps are eliminated during the queries. During each step, the model file is updated such that the calculations that were done for the earlier queries can be used for later queries.

Listing 4: Sample satisfyaliblity query

```
1 read_image simpleAES.model
2 add_atpg_constraints a 1 -module simpleAES t0
3 add_faults -stuck 1 N13
4 set_atpg -full_seq_time {120 360}
5 run_atpg
```

We model all satisfiability queries as a combination of *"atpg_constraints"* and *"stuck-at_faults"*. Algorithm 6 presents the steps involved in constructing the satisfiability

graph. First, we go through all the rare events selecting one at a time. Then, we prepare the ATPG script by implementing the selected rare event as an output constraint. Next, we implement all the remaining rare events as stuck-at faults and feed them to the ATPG tool. Since we have the instrumented design with all the rare events connected to the output, we have the observability of all the rare events. Therefore, we observe the rare events in the output to identify the activated faults. Listing 4 shows an example query for constructing the satisfiability graph of the example design provided in Listing 1. We implement a timeout interval to prevent graph creation from getting stuck in one query. Depending on the processing capability of the system that hosts the ATPG tool, the timeout would be different.

Our proposed method only takes **n** queries to construct the graph when we have **n** nodes, while TARMAC [3] makes $n \times \frac{n-1}{2}$ independent queries to construct the satisfiability graph. Although the ATPG tool calculates the connectivity between all the nodes, the extra overhead of each query can be reduced significantly with our proposed method. Figure 4 presents the comparison of the graph generation time for TARMAC [3] including lazy construction with our proposed MC-ATPG. It can be observed that the proposed method drastically reduces the graph construction time.

Algorithm 6 getSATisfiabilityGraph()

Input InstrumentedDesign D, rareEvents R_e
Output graph g

for Each $r_i \in R_e$ **do**
  tcl $\leftarrow \emptyset$
  tcl.add(r_i as a $po\_contraint$)
  tcl.add($\forall(R_e - r_i)$ as $stuck\text{-}at\_faults$)
  for $po \in$ OutputVector($ATPG(D, tcl)$) **do**
    if $po \in R_e$ and $po == 1$ **then**
      $e \leftarrow edge(r_i, po)$
      $g.add(e)$
    end if
  end for
end for
Return g

B. Maximal Clique Partitioning

Finding and listing all the cliques is an NP-complete problem. While there are promising algorithms, such as the Bron Kerbosch algorithm, the time complexity is in the order of $O(3^{\frac{n}{3}})$. Therefore a more quick approximation method is preferable to reduce the time taken for clique partitioning. We use the algorithm proposed by Eppste et al. with the worst-case time in the order of $O(dn3^{\frac{d}{3}})$ [23], where d is the smallest number such that every sub-graph of the original graph contains a vertex of at most degree of d. An example maximal clique that we can identify from the satisfiability graph in Figure 8 is the sub-graph consisting of three nodes A, B and C. Once all the maximal cliques are identified, we sort them by the clique size and pass to the next step to generate test vectors to activate each of them.

C. Maximal Clique Activation using ATPG

All the maximal cliques that were identified in Section VII-B are queried from the ATPG tool. Algorithm 7 presents the steps involved in the final clique activation step. Here we list all the vertices in the clique as *stuck-at-faults* and *po_constraints* and make the ATPG query. The ATPG tool will provide a test vector for each maximal clique produced by Section VII-B. For example, the ATPG will generate the test vector '110100' (or '111000') to activate the maximal clique (ABC) in Figure 8.

Algorithm 7 activateClique()

Input InstrumentedDesign D, clique c
Output testVectors T

for Each $v \in c$ **do**
  tcl $\leftarrow \emptyset$
  tcl.add(v as a $po\_contraint$)
  tcl.add(v as a $stuck\text{-}at\_fault$)
  $T \leftarrow T \cup ATPG(D, tcl)$
end for
Return T

VIII. Coverage Analysis

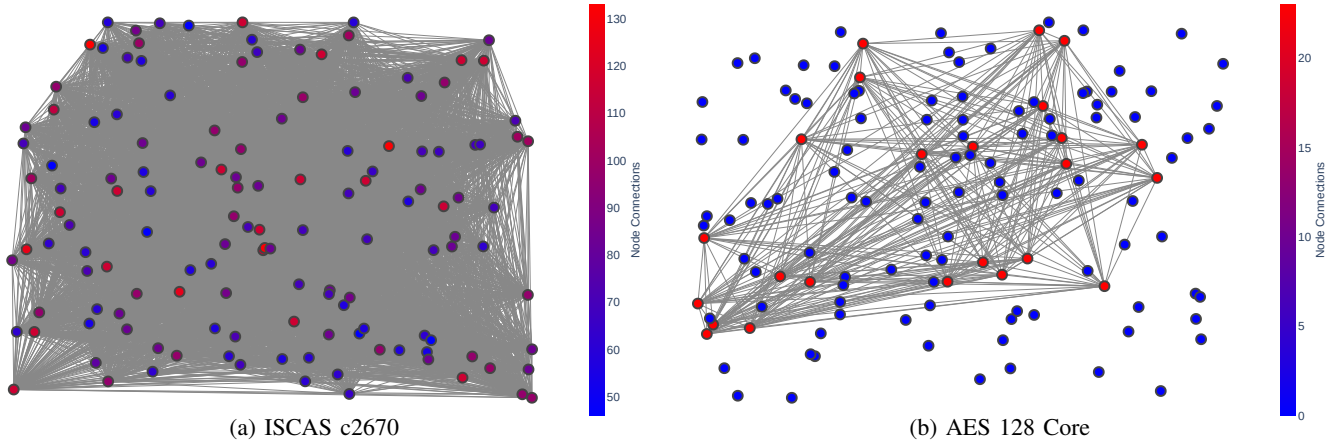
The generated test vectors should be evaluated for their effectiveness in detecting hardware Trojans. Algorithm 8 shows the three major steps in our coverage analysis. First, we construct the Trojan-infected benchmarks in two ways. (1) We used the automated Trojan insertion framework developed by Cruz et al. [24] that embeds a wide variety of Trojan configurations based on user constraints such as the type of Trojan, Trojan activation probability, number of triggers, and choice of payload. The tool also ensures that the inserted Trojan is valid. (2) We have also used Trojan inserted AES benchmarks from Trust-Hub [25]. Note that our Trojan insertion framework is independent of our test generation framework to ensure fair evaluation of our test generation techniques. Next, we simulate both golden and Trojan inserted benchmarks using the generated test patterns. Any mismatch in the outputs indicates a successful Trojan detection. Finally, the effectiveness of the generated tests are calculated as a percentage of the number of detected Trojans (activated triggers) over the total number of Trojans (valid triggers) in the benchmark suite.

IX. Experiments

The section demonstrates the effectiveness of our test generation framework. First, we describe the experimental setup. Next, we present the experimental results.

A. Experimental Setup

We have developed scripts to automate the entire process. *Yosys* [26] synthesis tool is used to flatten the hierarchical designs before rareness analysis. For rareness and coverage analysis simulations, we have used *Synopsys VCS* simulator. For compiling the RTL designs to the gate-level netlist, *Synopsys DC Compiler* is used. We have used *Synopsys Tetramax*



(a) ISCAS c2670 (b) AES 128 Core

Fig. 9: Satisfiability graph constructed for ISCAS2670 [21] and AES 128 Core [22] using the proposed method. In order to keep the number of vertices the same for fair comparison, we have selected the rarest 130 nodes from both designs. It can be observed that ISCAS benchmarks produce almost complete graphs while real-world IPs, such as AES, produce spares graphs. The lazy construction technique used by TARMAC [3] works well with ISCAS benchmarks. However, in case of real-world designs, TARMAC's performance is negatively impacted during maximal clique activation as demonstrated in Section IX-E.

TABLE II: Effectiveness of Trojan and trigger coverage compared with other techniques. (Rareness threshold: 0.2, Number of rare signals in a Trigger varies between 1 to 6, Sample set size: 1000, N: 1000)

Design	# of Gates	# of Rare Signals	Trojan Coverage (%)				
			MERO [2]	TARMAC [3]	TGRL [9]	ND-ATPG	MC-ATPG
c2670	1269	299	27.6	87.8	92.2	91.6	100
c5315	2307	443	49.3	80.1	95.2	94.7	100
c6288	2416	479	57.2	76.7	100	100	100
c7552	3513	517	39.1	47.1	99.1	100	100
s13207	2573	792	8.2	71.3	92.3	93.1	100
s15850	3448	897	11.7	66.5	87.5	90.2	100
s35932	12204	1027	15.2	80.4	89.8	89.7	100
Average	3962	636.3	29.8	72.8	93.7	94.2	100.0

Algorithm 8 Coverage Analysis

Input Design D, Tests
Output coverage $M\%$

1: $\{D_{TR}, N_{TR}\} \leftarrow insertTrojans(D)$ + other sources
2: **for** each $t_i \in Tests$ **do**
3: $X \leftarrow simulate(D, t_i)$
4: $Y \leftarrow simulate(D_{TR}, t_i)$
5: **if** $X \neq Y$ **then**
6: $activated++$ ▷ $activated$ was initialized to 0
7: **end if**
8: **end for**
9: Return $(activated/N_{TR}) \times 100$

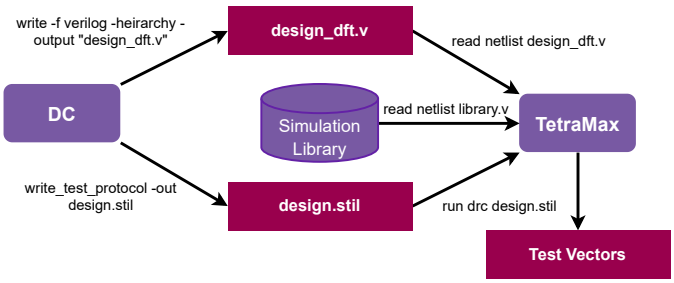


Fig. 10: Overview of our framework for synthesis with Synopsys Design Compiler (DC) and test pattern generation with Synopsys TetraMax.

as the ATPG tool. Figure 10 illustrates the steps involved in synthesizing and test generation with *DC Compiler* and *TetraMax*. For maximal clique partitioning in near-optimal time, *iGraph* [27] graph library is used. For validating the sampled Trojan triggers, EBMC [28] model checker was used. All the experiments including the execution of state-of-the-art test generation methods were carried out in a server environment with Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz processor and 64GiB Memory. To provide an advantage to the state-of-the-art methods, we utilize designs without a scan chain. We also discuss in Section IX-F the potential for drastic improvement in test generation time and Trojan coverage for designs with scan chains.

Since most of the state-of-the-art test generation platforms have shown their effectiveness on ISCAS [21] benchmarks, we have used ISCAS benchmarks to show the effectiveness of the proposed approach. We have also used Advanced Encryption Standard (AES), Elliptic Curve Digital Signature Algorithm (ECDSA) cryptographic designs, and several processor cores developed for RISC-V instruction set architecture to show the scalability of our approach. We have collected the Trojan-inserted benchmarks following the procedure outlined in Section VIII. We compare the following five approaches.

- **ND-ATPG**: Our proposed approach for ATPG-based N-activation of rare events (Section VI).

- **MC-ATPG**: Our proposed approach for ATPG-based maximal clique activation (Section VII).
- **MERO** [2]: State-of-the-art statistical test generation based Trojan detection (Section II).
- **TARMAC** [3]: State-of-the-art maximal clique sampling based Trojan detection (Section II).
- **TGRL** [9]: State-of-the-art statistical test generation using reinforcement learning (Section II).

The original versions of MERO [2], TARMAC [3] and TGRL [9] did not support RTL models since they were designed specifically focusing on ISCAS netlist format. For the scalability evaluation, we have modified their netlist parsers to process Verilog designs. The implementation of TGRL [9] is based on reinforcement learning that computes SCOAP parameters using SAT solvers. We could not modify the implementation (designed for ISCAS netlist format) to support Verilog RTL models. As a result, we are not able to compare scalability with TGRL. Since TGRL is based on MERO, which has fundamental scalability limitations (as discussed next), we believe TGRL will also inherit the same scalability concerns.

TABLE III: Elimination of false positive rare signals.

Design	Signals in Flattened Net-List				Reduction
	Total Nets	Rare	False Rare	True Rare	
RV-UE	497843	24545	5064	19481	20.6%
RISCY	352968	13496	3609	9887	26.7%
ECC	107925	492	107	385	21.7%
AES	90956	371	70	301	18.8%
RSA	1807	74	4	70	5.4%

B. Elimination of False Positive Rare Signals

We observed that most of the RISC-V processor designs and cryptographic designs contain a considerable amount of fixed value signals. The rare signals were calculated after 10,000 random simulation iterations for the respective pipeline depth of the design and filtered with the rareness threshold of 0.01. Table III shows the reduction of false rare signals based on static analysis. The first column indicates the design under consideration. The second and third columns show the number of total and rare signals, respectively. The fourth column shows the number of false rare signals. The next column shows the number of true rare signals after removing the false rare signals. The last column shows the percentage reduction due to the removal of the false rare signals. The table illustrates that structural analysis removes a considerable percentage (up to 26%) of false positive rare signals. This can significantly reduce the test generation time since it has to deal with less number of signals. Moreover, the reduction of false rare signals reduces the chances of constructing invalid Trojan triggers during the evaluation of Trojan coverage.

C. Comparison with State-of-the-art Methods

Table II shows the results for combinational and sequential ISCAS benchmarks. For all the methods, the rareness threshold was selected as 0.2. For fair evaluation, during coverage analysis, we have considered the same set of sampled and validated triggers on all the methods. Clearly, MC-ATPG and ND-ATPG significantly outperform MERO and TARMAC, and provide comparable/better results than TGRL. Note that the existing approaches (including TGRL which is built on top MERO) are not scalable as demonstrated in the next section.

D. ND-ATPG versus MERO

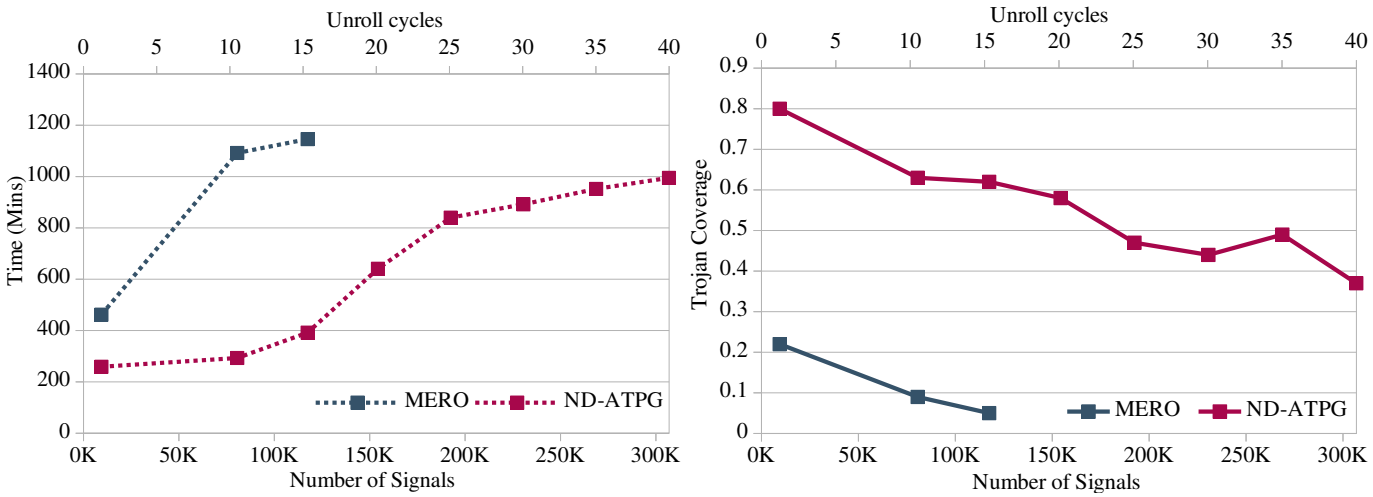
In order to demonstrate the scalability of ND-ATPG, we have created two experiments. In the first experiment, we used the cryptographic AES core. We unrolled the AES design for various numbers of cycles such that it increases the number of signals in the design. Figure 11a illustrates the time taken to generate test vectors with ND-ATPG compared with MERO while Figure 11b illustrates the Trojan coverage of ND-ATPG compared with MERO on cycle unrolled AES benchmarks. In the second experiment, we used RISC-V processor cores implemented in Verilog with the design complexity (number of signals in the flattened netlist) ranging from about 3K to 500K signals as shown in Figure 12a. Figure 12b and Figure 12c present the comparison between ND-ATPG versus MERO for test generation time and observed Trojan coverage, respectively. The results of both experiments reveal that the proposed method is scalable on large designs.

We did not compare ND-ATPG with TARMAC since TARMAC failed to run even on the smallest AES design (AES unrolled for 1 cycle). This is due to the fact TARMAC algorithm struggles at the logical expression generation stage as well as the SAT-solving stage due to the large number of signals in the design. MERO on the other hand has a feedback loop, where it does a bit flip and monitors for the rare signal activation. When the number of rare signals are high, this monitoring process gets tedious and eventually MERO fails. It can be seen in Figure 11 and Figure 12 that MERO fails when the number of signals in the design reaches more than 100K during both experiments. Our proposed method performs well on complex designs while maintaining high Trojan coverage.

In order to compare the efficiency of MERO and ND-ATPG on the AES benchmark, we analyzed the Trojan coverage with two methods once each method produces a test vector. Figure 13 demonstrates that due to the forced activation of each rare signal by ND-ATPG, coverage increases within a small time duration. However, the Trojan coverage given by MERO remains inferior.

E. MC-ATPG versus TARMAC

Table IV shows the coverage results for the benchmarks of AES, ECDSA, and two RISC-V processor cores. For coverage calculations, we used varying trigger sizes from 1-10. When the design complexity is high, TARMAC gets stuck at the logical expression generation stage and fails the instrumentation stage. As expected, TARMAC failed to create satisfiability queries for complex designs of AES, ECDSA, RISCY, and RISCV-UE. Therefore, to make a fair comparison we have included sub-modules corresponding to each benchmark. We flattened the designs selecting sub-modules as a top module and obtained results for Sbox, SboxX, eeageneric, MontgomerryLadder, Execute, Decode, Issue, and LSU of the corresponding benchmarks. For all the experiments under this section, we selected the rarest 130 signals from the selected design. Due to the lazy construction of the satisfiability graph



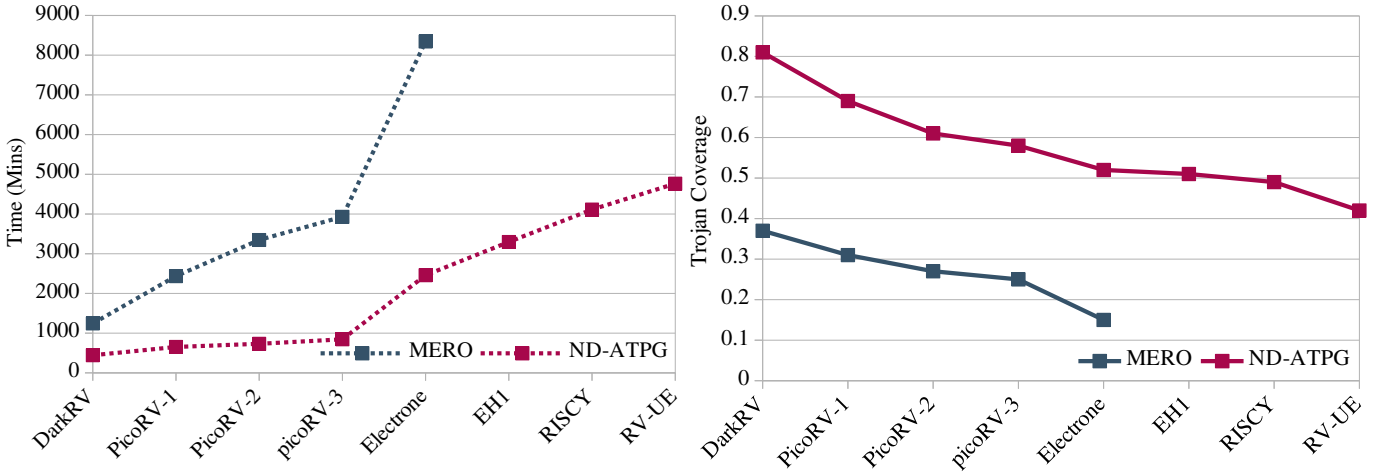
(a) Time taken for cycle unrolled AES

(b) Trojan coverage observed for cycle unrolled AES

Fig. 11: Scalability of ND-ATPG compared with MERO for cycle unrolled AES designs: (a) test generation time, and (b) hardware Trojan coverage.

(a) RISC-V Core sizes with respect to the number of signals in the flattened design.

RISCV Core	DarkRV	picoRV-1	picoRV-2	picoRV-3	Electrone	EH1	RISCY	RV-UE
No. Signals	3806	6437	8632	15864	104232	194637	352968	497843



(b) Time taken for different RISC-V cores

(c) Trojan coverage observed for different RISC-V cores

Fig. 12: Scalability of ND-ATPG compared with MERO for different RISC-V processor cores: (a) eight RISC-V designs with increasing complexity, (b) test generation time, and (c) Trojan coverage.

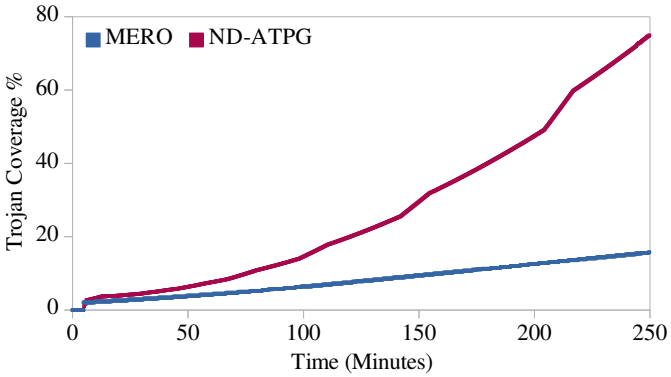


Fig. 13: Trojan coverage of MERO versus ND-ATPG on the tests generated over same duration for AES Benchmark (for Trojans with trigger size 1/2/3/4)

in TARMAC, there are an exponential number of false cliques in the satisfiability graph. To address the exponential problem, TARMAC employs random sampling of cliques. As a result, it misses the legitimate maximal cliques, which can significantly affect the Trojan coverage. However, MC-ATPG constructs the complete satisfiability graph with ATPG queries explained in Section VII-A, and therefore, MC-ATPG does not generate any false cliques. Since our approach can be computed in parallel, the test generation time can be drastically reduced by performing parallel computation of ATPG queries. Moreover, the Trojan coverage can be improved by increasing the ATPG query time limit.

F. Comparison of Rare Branch Activation with ND-ATPG

In order to demonstrate the effectiveness of the proposed approach on rare branches, a separate experiment was designed.

TABLE IV: Scalability of MC-ATPG compared with TARMAC for IP level Trojan detection considering rarest 130 signals with varying trigger sizes from 1-10 on several cryptographic IPs and CPU cores. Here it can be observed that with lazy construction TARMAC takes a longer time to finish due to invalid clique activation queries from the SAT solver.

Design	Module	No. of Signals	TARMAC				MC-ATPG			
			Time (Mins)			Cov %	Time (Mins)			Cov %
			Graph Generation	Clique Partitioning	Clique Activation		Graph Generation	Clique Partitioning	Clique Activation	
AES	*Sbox*	2056	172.76	22.83	491.45	82.1	39.53	0.05	22.37	100
	SboxX	2056	193.01	16.98	647.01	79.2	42.43	0.05	29.43	100
	AES (Top)	90956	Instrumentation Failed				3768.09	0.09	446.88	91.2
ECDSA	*eeageneric*	1936	183.55	17.96	531.29	85.4	30.22	0.05	17.73	99.7
	montgomeryLadder	20356	Instrumentation Failed				1897.38	0.09	884.62	87.8
	ECDSA (Top)	107925	Instrumentation Failed				2984.89	0.11	2002.11	82.3
RISCY	*Execute*	3723	36.34	5.63	44.3	88.5	16.71	0.01	14.32	100
	Decode	13833	205.32	24.64	638.86	67.9	115.32	0.23	78.65	74.54
	Core (Top)	352968	Instrumentation Failed				4786.98	0.56	2967.89	59.09
RISCV-UE	*Issue*	5642	132.98	16.1	397.3	89.73	43.73	0.01	43.76	95.89
	CSR	10977	196.5	16.8	484.53	73.43	79.03	0.06	74.08	79.04
	Core (Top)	497843	Instrumentation Failed				4912.89	0.76	4143	54.78

Table V shows the results generated from Trojan inserted AES benchmarks from Trust-Hub [25]. These AES benchmarks are specifically designed for side-channel analysis and triggers are embedded in rare branches. We have generated results on both scan-chain and non-scan versions of the same benchmark and compared them with MERO. ND-ATPG outperforms MERO due to the fact that ND-ATPG forcefully activates the rare branches while MERO relies on bit flipping of the initial random vectors. Also, when we consider ND-ATPG, scan-chain versions can reach 100% branch coverage quickly. This also validates the fact that statistical test generation (e.g., MERO) designed for activating rare signals are not suitable for activating rare branches.

TABLE V: Rare branch activation effectiveness of MERO compared with ND-ATPG on scan and non-scan designs

Design	MERO [2]		ND-ATPG			
			Non-scan		With Scan	
	Branch Cov%	Time (S)	Branch Cov%	Time (S)	Branch Cov%	Time (S)
AES_T300	0%	54000	75%	50401	100%	8.2
AES_T400	0%	54000	40%	46400	100%	7.1
AES_T500	0%	54000	28%	16980	100%	8.2
AES_T600	8%	54000	41%	59801	100%	11
AES_T700	0%	54000	27%	27200	100%	8.3
AES_T800	0%	54000	44%	38405	100%	1.9
AES_T900	33%	54000	100%	15435	100%	3.2
AES_T1000	0%	54000	34%	46601	100%	6.7
AES_T1100	0%	54000	20%	47109	100%	5.2
AES_T1200	0%	54000	33%	59026	100%	1.8

G. N-Activation of ND-ATPG versus MERO

N-activation of ND-ATPG and N-Detection of MERO are similar in concept but there is a considerable difference in the performance. In order to demonstrate this, we have selected an ECDSA module and observed the activation of rare events. Figure 3 shows the results of the experiment. MERO was unable to activate most of the rare signals even once, while less rare signals got activated more than N times (leading to inferior Trojan coverage). However, ND-ATPG (N-activation) has forced to activate almost all the rare events almost N times evenly which leads to superior Trojan coverage.

X. Applicability and Limitations

In this section, we discuss the applicability and limitations of the proposed test generation algorithms. Specifically, we discuss four aspects: (1) practical considerations during rareness threshold selection, (2) detection of diverse Trojan types, (3) the requirement of controllability for the effectiveness of the proposed techniques, and (4) applicability beyond hardware Trojan detection.

A. Rareness Threshold Selection

Our test generation approach consists of two parts: (i) rareness analysis to produce a set of rare (target) signals and (ii) test generation to maximize the likelihood of detecting a Trojan constructed using the target signals. The rareness threshold should be chosen such that the selected rare signals cannot be activated by traditional simulation using millions of random tests. These are the signals that an attacker is likely to use to construct a trigger that can stay hidden during traditional validation. There are several practical considerations since there can be thousands of rare signals in a typical million-gate design that cannot be activated during traditional validation. If we set the rareness threshold too low, the number of rare signals in the set will be less, and the test generation algorithm will finish faster. However, the attacker may use other rare signals (outside the set). If we set the rareness threshold too high, the test generation will take more time but we can cover more scenarios where an attacker can mount a Trojan.

There can be scenarios when the rareness threshold may be partially or completely irrelevant. For example, a designer may bypass rareness analysis and provide a list of suspicious signals as input for test generation. Similarly, a designer may choose a low rareness threshold to select very rare signals and manually add other signals based on specific objectives (e.g., XOR-LFSR related signals as discussed next).

B. Detection of XOR-LFSR Trojans

We evaluated the applicability of our approach on diverse benchmarks in ISCAS and Trust-Hub. Each of these Trojans relies on the assumption that an attacker is likely to construct a trigger using rare events. In other words, the first step of our framework performs rareness analysis to construct a set of target (rare) events, which is used as an input by our proposed test generation algorithms (ND-ATPG and MC-ATPG) to maximize the likelihood of detecting a

Trojan constructed using the target events. The test generation algorithms proposed in this paper are also applicable when the above assumption is not true. For example, consider a XOR-LFSR [29] Trojan that is constructed from Linear Feedback Shift Registers (LFSR). To detect XOR-LFSR Trojans, we need to select the signals from the LFSR as target signals (instead of rare signals or in addition to rare signals). Note that the test generation algorithms do not require any changes. LFSR can be implemented in two ways - using an initial seed value based function or an internal fixed value based function (not likely to be used by an attacker due to the predictability of the LFSR output). In case of initial seed value-based functions, our approach is effective since it is utilizing statistical guarantees of activating each register bit N times, and therefore, it increases the likelihood of activating the Trojan trigger irrespective of which bits are selected by the adversary to construct the XOR gate. If the inputs cannot control the LFSR, our proposed approach performs comparable to random tests since there is no controllability over the LFSR register.

C. Controllability Considerations

Test generation based techniques rely on the controllability of the internal signals to generate effective test cases. In the case of ND-ATPG, the reachability of N activation for every signal is determined by the controllability of signals in the design. If a design does not have good controllability, there will be more signals that do not have N possible tests (even if we try all possible input combinations). Figure 14 shows how many times a signal got activated by ND-ATPG and MERO for 1000 rare signals in ECC_Cordinate_conversion module in ECDSA benchmark with N as 100. As our experimental results demonstrate, we are able to activate a vast majority of the signals N times. In fact, the remaining ones got activated very close to N times. For example, the tallest brown bar indicates that about 780 signals got activated 100 times by ND-ATPG. In contrast, only a small percentage of signals get activated N times by state-of-the-art (MERO [2]). For example, the tallest blue bar indicates that about 340 signals (out of 1000) were not activated at all by MERO. Note that the summation of the heights of all the bars in the same color should be 1000 (total number of rare signals).

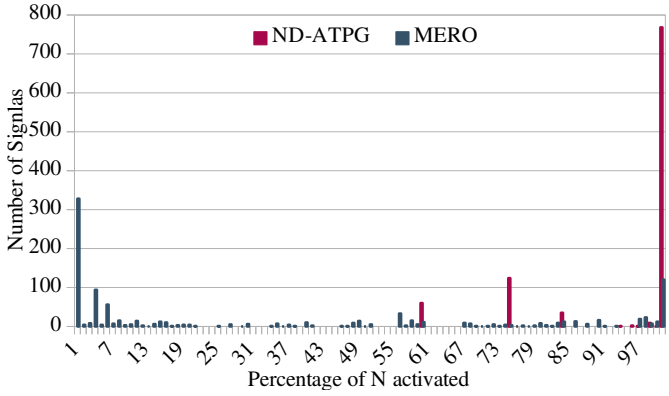


Fig. 14: Number of signals that are activated N=100 times for MERO versus ND-ATPG. Due to the directed nature of tests, ND-ATPG is able to activate signals close to N times.

D. Applicability beyond Trojan Detection

Although our primary focus in this paper is hardware Trojan detection, the algorithms proposed in this paper can be utilized for security validation, functional validation, manufacturing testing, as well as reliability validation. Specifically, the ATPG-based scalable test generation algorithms are effective in activating a specific node N times (ND-ATPG) or activating M nodes at the same time (MC-ATPG). Therefore, ND-ATPG can be utilized to improve functional coverage by activating corner cases and hard-to-activate functional behaviors [30]. Similarly, ND-ATPG can improve trust coverage by activating hard-to-detect security vulnerabilities [31]. Likewise, MC-ATPG can be adopted in manufacturing testing to generate efficient test vectors to activate multiple stuck-at-faults [32].

XI. Conclusion

Hardware Trojan detection is a major challenge due to the difficulty in activating rare triggers. While there are many promising test generation approaches for Trojan detection, they are not scalable for large designs. Moreover, their applicability is limited to detecting specific types of Trojans. In this paper, we proposed an ATPG-based scalable framework for activation of rare events consisting of both rare signals and rare branches. This paper made two important contributions. We show that both rare signals and rare branches can be mapped to stuck-at faults. As a result, ATPG can be used to generate test patterns to activate rare events. We have developed an automated constraint generation method to perform N-activation of rare events as well as maximal clique coverage of rare events using ATPG. Our experimental results demonstrated that our proposed approach significantly outperforms the state-of-the-art test generation methods. It highlights the fact that existing statistical approaches are not effective in satisfying the N-detect principle and therefore provide inferior Trojan coverage compared to our approach. It also reveals that the existing test generation techniques for activating rare signals are not suitable for activating rare branches, highlighting the need for our proposed approach.

Acknowledgments

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

References

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

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

[3] Y. Lyu and P. Mishra, "Scalable activation of rare triggers in hardware trojans by repeated maximal clique sampling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 7, pp. 1287–1300, 2021.

[4] 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, 2015. [Online]. Available: https://doi.org/10.1109/TCAD.2015.2422836

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

[6] 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*, 2008, pp. 1362–1365.

[7] H. D. Foster, "Trends in functional verification: A 2014 industry study," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.

[8] C.-P. Wen, L.-C. Wang, and K.-T. Cheng, "Simulation-based functional test generation for embedded processors," *IEEE Transactions on Computers*, vol. 55, no. 11, pp. 1335–1343, 2006.

[9] Z. Pan and P. Mishra, "Automated test generation for hardware trojan detection using reinforcement learning," in *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2021, pp. 408–413.

[10] Y. Huang, S. Bhunia, and P. Mishra, "MERS: Statistical test generation for side-channel analysis based trojan detection," in *ACM Conference on Computer and Communications Security*, 2016, p. 130–141.

[11] S. Narasimhan, X. Wang, D. Du, R. S. Chakraborty, and S. Bhunia, "Tesr: A robust temporal self-referencing approach for hardware trojan detection," in *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*. IEEE, 2011, pp. 71–74.

[12] S. Narasimhan, D. Du, R. S. Chakraborty, S. Paul, F. G. Wolff, C. A. Papachristou, K. Roy, and S. Bhunia, "Hardware trojan detection by multiple-parameter side-channel analysis," *IEEE Transactions on computers*, vol. 62, no. 11, pp. 2183–2195, 2012.

[13] Z. Shi, H. Ma, Q. Zhang, Y. Liu, Y. Zhao, and J. He, "Test generation for hardware trojan detection using correlation analysis and genetic algorithm," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 4, pp. 1–20, 2021.

[14] C. Nigh and A. Orailoglu, "Adatrust: Combinational hardware trojan detection through adaptive test pattern construction," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 3, pp. 544–557, 2021.

[15] M. Banga and M. S. Hsiao, "A region based approach for the identification of hardware trojans," in *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*. IEEE, 2008, pp. 40–47.

[16] R. Elnaggar, K. Chakrabarty, and M. B. Tahoori, "Hardware trojan detection using changepoint-based anomaly detection techniques," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 12, pp. 2706–2719, 2019.

[17] M. Banga and M. S. Hsiao, "Trusted rtl: Trojan detection methodology in pre-silicon designs," in *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2010, pp. 56–59.

[18] Y.-T. Lin, C. U. Ezekwe, and R. D. Blanton, "Physically-aware n-detect test relaxation," in *2009 27th IEEE VLSI Test Symposium*, 2009, pp. 197–202.

[19] G. Chen, J. Rajski, S. Reddy, and I. Pomeranz, "N-distinguishing tests for enhanced defect diagnosis," in *2009 Asian Test Symposium*, 2009, pp. 183–186.

[20] J. Cruz, F. Farahmandi, A. Ahmed, and P. Mishra, "Hardware trojan detection using atpg and model checking," in *International Conference on VLSI Design (VLSID)*, 2018, pp. 91–96.

[21] "ISCAS89." [Online]. Available: pld.ttu.ee/˜maksim/benchmarks/

[22] "Aes128." [Online]. Available: http://opencores.org/projects/apbtoaes128

[23] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in sparse graphs in near-optimal time," 2010.

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

[25] "Trusthub," https://www.trust-hub.org/, accessed: 2021-9-10.

[26] W. Clifford, "Yosys open synthesis suite," http://www.clifford.at/yosys/, 2013.

[27] G. Csardi and T. Nepusz, "The igraph software package for complex network research," *InterJournal*, vol. Complex Systems, p. 1695, 2006. [Online]. Available: https://igraph.org

[28] R. Mukherjee, D. Kroening, and T. Melham, "Hardware verification using software analyzers," in *ISVLSI*, 2015.

[29] S. K. Haider, C. Jin, M. Ahmad, D. M. Shila, O. Khan, and M. van Dijk, "Advancing the state-of-the-art in hardware trojans detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 1, pp. 18–32, 2017.

[30] M. Chen, X. Qin, H.-M. Koo, and P. Mishra, *System-level validation: high-level modeling and directed test generation techniques*. Springer Science & Business Media, 2012.

[31] F. Farahmandi, Y. Huang, and P. Mishra, *System-on-Chip Security*. Springer, 2020.

[32] M. Fujita and A. Mishchenko, "Efficient sat-based atpg techniques for all multiple stuck-at faults," in *2014 International Test Conference*. IEEE, 2014, pp. 1–10.

Aruna Jayasena is a Ph.D student in the Department of Computer & Information Science & Engineering at the University of Florida. He received his B.S. in the Department of Computer Science and Engineering at the University of Moratuwa, Sri Lanka, in 2019. His research focuses on hardware security, test generation, and system-on-chip debug.

Prabhat Mishra is a Professor in the Department of Computer and Information Science and Engineering and a UF Research Foundation Professor at the University of Florida. He received his Ph.D. in Computer Science from the University of California at Irvine in 2004. His research interests include embedded systems, design automation, hardware security, energy-aware computing, formal verification, system-on-chip validation, machine learning, and quantum computing. He currently serves as an Associate Editor of IEEE Transactions on VLSI Systems and ACM Transactions on Embedded Computing Systems. He is an IEEE Fellow, an AAAS Fellow, and an ACM Distinguished Scientist.