MaxSense: Side-channel Sensitivity Maximization for Trojan Detection Using Statistical Test Patterns

YANGDI LYU and PRABHAT MISHRA, University of Florida

Detection of hardware Trojans is vital to ensure the security and trustworthiness of System-on-Chip (SoC) designs. Side-channel analysis is effective for Trojan detection by analyzing various side-channel signatures such as power, current, and delay. In this article, we propose an efficient test generation technique to facilitate side-channel analysis utilizing dynamic current. While early work on current-aware test generation has proposed several promising ideas, there are two major challenges in applying it on large designs: (i) The test generation time grows exponentially with the design complexity, and (ii) it is infeasible to detect Trojans, since the side-channel sensitivity is marginal compared to the noise and process variations. Our proposed work addresses both challenges by effectively exploiting the affinity between the inputs and rare (suspicious) nodes. The basic idea is to quickly find the profitable ordered pairs of test vectors that can maximize side-channel sensitivity. This article makes two important contributions: (i) It proposed an efficient test generation algorithm that can produce the first patterns in the test vectors to maximize activation of suspicious nodes using an SMT solver, and (ii) it developed a genetic-algorithm based test generation technique to produce the second patterns in the test vectors to maximize the switching in the suspicious regions while minimizing the switching in the rest of the design. Our experimental results demonstrate that we can drastically improve both the side-channel sensitivity (62× on average) and time complexity (13× on average) compared to the state-of-the-art test generation techniques.

CCS Concepts: • **Hardware → Test-pattern generation and fault simulation**; • **Security and privacy → Side-channel analysis and countermeasures**;

Additional Key Words and Phrases: Hardware Trojan detection, test generation, dynamic current, side-channel analysis

ACM Reference format:
Yangdi Lyu and Prabhat Mishra. 2020. MaxSense: Side-channel Sensitivity Maximization for Trojan Detection Using Statistical Test Patterns. *ACM Trans. Des. Autom. Electron. Syst.* 26, 3, Article 22 (January 2021), 21 pages.
https://doi.org/10.1145/3436820

1 INTRODUCTION

Hardware Trojans are malicious modifications incorporated during the System-on-Chip (SoC) design cycle [13, 22, 27, 35, 48, 52]. As IC design and fabrication process becomes more and more globalized, the threat of hardware Trojan attacks is increasing due to potential malicious modifications at different stages of the design and fabrication process [13, 17, 20, 21]. To ensure

This work was partially supported by the National Science Foundation (NSF) grant CCF-1908131.
Authors' address: Y. Lyu and P. Mishra, University of Florida, Gainesville, FL, 32611, USA; emails: lvyangdi0729@gmail.com, prabhat@ufl.edu.

1084-4309/2020/01-ART22 $15.00
https://doi.org/10.1145/3436820

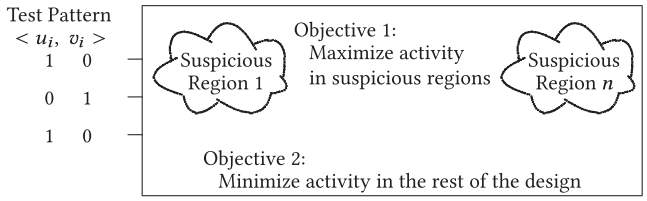


Fig. 1. The goal of our framework is to generate test patterns that can maximize switching in the suspicious regions (objective 1) while minimizing switching in the rest of the design (objective 2) to significantly improve the side-channel sensitivity. The figure shows an example pair of test patterns (<101,010>).

trustworthy SoC design, it is crucial to develop efficient techniques for detection and localization of such malicious implants. There is a wide variety of approaches for hardware Trojan detection using a combination of formal methods [5, 18, 20, 31, 34], test generation for trust validation [10, 12, 14, 39, 49–51, 53], and side-channel analysis [6, 11, 16, 24, 26, 28, 29, 37, 38, 40, 41, 43, 44, 47]. Recent efforts have explored a profitable combination of various approaches for Trojan detection. For example, test generation has been utilized for effective side-channel analysis [24].

Simulation-based approaches are the most widely used form of security validation today. There are several test generation efforts for detection of hardware Trojans [7, 14, 24, 30, 32, 33]. The existing test generation approaches can be broadly categorized as logic testing and side-channel analysis. Side-channel analysis does not require the Trojan to be fully activated or to propagate its effect to the observable outputs. However, detection of small Trojans can be hard, since the change in side-channel signatures can be negligible compared to the environmental noise or process variations. Logic testing is immune to the noise and process variations, but requires both the activation of the Trojan and propagation of the Trojan effects to the observable outputs. Since the number of possible input patterns is exponential [15], Trojan detection using logic testing can be infeasible for large designs. While MERS [23, 24] tried to combine the advantages of logic testing and side-channel analysis, there are two major challenges in applying it on large designs. The test generation time using MERS grows exponentially with the design complexity. Moreover, it is infeasible to detect Trojans, since the increase in side-channel sensitivity is marginal compared to the noise and process variations. Specifically, MERS typically achieves less than 2% sensitivity whereas process variations can be more than 10% [9]. Our proposed approach addresses both challenges.

This article introduces an efficient approach, referred to as MaxSense, to generate test patterns to maximize the side-channel sensitivity for Trojan detection. *In this article, we target the dynamic current as our side-channel signature.* However, this approach can also be extended to the other side-channel parameters with suitable modifications of the evaluation criterion. The main idea of MaxSense is to find a set of ordered pairs of test patterns, $T = \{t_i\} = \{<u_i,v_i>\}$, to realize two objectives when applying them to the design, as shown in Figure 1. We refer t_i as an ordered pair of test patterns with u_i as its first pattern and v_i as its second pattern. The first pattern in the ordered pair (u_i) tries to maximize the activation of the suspicious regions. The second pattern (v_i) needs to simultaneously satisfy two objectives. The first objective is to maximize switching in the suspicious regions. The second objective is to minimize the switching in the rest of the design, such that the side-channel sensitivity is maximized. As demonstrated in Section 3.3, the selections of both u_i and v_i are equally important to enable efficient test generation for effective side-channel analysis. Specifically, this article makes the following major contributions:

- Exploits the input affinity to identify test patterns that can maximize switching in the suspicious (target) regions while minimizing switching in the rest of the circuit to significantly improve the side-channel sensitivity.
- Proposes a fast and effective Satisfiability Modulo Theories (SMT)-based approach to increase the activation probability in the suspicious regions.
- Utilizes genetic algorithm to quickly find the profitable test patterns by exploiting affinity of the inputs to the suspicious regions, thus improving the side-channel sensitivity.
- The significant improvement in sensitivity enabled MaxSense to detect the majority of Trojans (out of randomly inserted 1,000 Trojans), while the state-of-the-art approaches can detect less than 2% Trojans.

The article is organized as follows: Section 2 describes existing Trojan detection techniques. Section 3 provides problem formulation and motivates the need for our work. Section 4 describes our test generation framework. Section 5 presents experimental results. Section 6 concludes the article.

2 RELATED WORK

Logic testing and side-channel analysis are two of the primary approaches to detect hardware Trojans. In this section, we outline the related efforts in these two techniques and genetic algorithm.

2.1 Logic Testing-based Trojan Detection

Logic testing [8, 12, 14, 42, 49–51, 53] in Trojan detection has been extensively explored. The MERO approach presented in Reference [14] utilized the idea of N-detect [8, 42] to achieve high coverage over their randomly sampled Trojans, assuming the trigger conditions of the Trojans consist of rare nodes only. The authors observed that if the generated test patterns are able to satisfy all rare values N times (N-detect criterion), it is highly likely that the unknown rare trigger conditions are satisfied when N is sufficiently large.

The framework of MERO (N-detect) [14] is shown in the left part of Figure 2. MERO is a constrained-random approach to achieve the N-detect criterion through flipping bits. It starts from a large number of random test patterns RT. Then, it simulates the netlist with each random test pattern and gets the number of rare values being satisfied (R_v). Next, these patterns are sorted based on R_v. The profitable test patterns (with greater R_vs) are visited earlier than the ones with smaller R_vs. For each random test pattern $rt_i \in RT$, each bit is flipped one after the other to improve its quality as shown in Algorithm 1. If the flipping of some bit can improve the N-detect criterion, the flipping is accepted. Otherwise, the flipping is reversed. The modified test pattern is put into the final test set if it is helpful in improving the N-detect criterion.

Since the N-detect criterion requires that each rare signal is activated by at least N times, this criterion is evaluated over all test patterns. Therefore, the checking of improvement on N-detect criterion (line 6 in Algorithm 1) needs to consider all the previously generated test patterns. This global evaluation method prevents N-detect approach from running in parallel. The time complexity of MERO depends on the number of random test patterns RT, the number of times that each rare signals should be satisfied (N), and the number of input signals (length of each test pattern). The larger N is, the more test patterns are required in RT to satisfy N-detect criterion. The total number of simulations is the multiplication of random vectors and the number of input signals.

2.2 Side-channel Analysis-based Trojan Detection

Logic testing approaches have several limitations such as a lack of scalability due to the long test generation time even for small benchmarks, the restrictions of the trigger conditions being fully

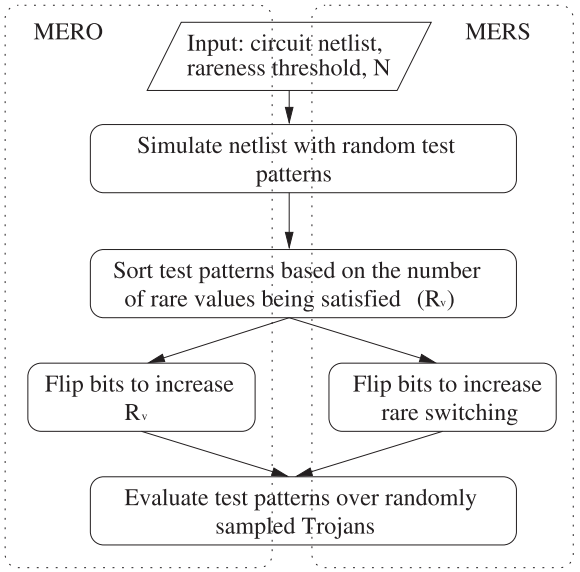


Fig. 2. MERO [14] versus MERS [24]. While MERS counts the number of rare switching, MERO counts the number of activated rare values.

ALGORITHM 1: MERO (N-detect) [14]

```
1:  procedure N − detect(random test patterns RT, N)
2:      for each rt_i in RT do
3:          for each b_i in rt_i do
4:              flip the bit b_i
5:              simulate the netlist with the modified rt_i
6:              if modified rt_i improves N-detect then
7:                  keep the flipping of b_i
8:              else
9:                  reverse the flipping of b_i
10:             end if
11:         end for
12:     end for
13: end procedure
```

activated, and the effect of the inserted Trojan propagating to the observable points. Side-channel analysis overcomes these disadvantages. Trojan detection using side-channel analysis [6, 11, 16, 24, 26, 28, 29, 37, 38, 40, 41, 43, 44, 47] measures transient current, power consumption, or path delay both in the golden design and the design under test. If the measured signals from these two designs vary by a threshold, a Trojan is suspected to be present.

Huang et al. [23, 24] extended the idea of N-detect test for side-channel analysis and proposed a test generation framework called MERS to maximize the sensitivity of dynamic current. The frameworks of MERS and MERO are similar as shown in Figure 2. MERS generates compact test patterns to let each rare node switch from its non-rare value to its rare value N times, increasing the probability of partially or fully activating a Trojan. After generating MERS test patterns, the authors proposed simulation-based reordering (called MERS-s) to decide the order of applying test patterns to maximize side-channel sensitivity. Although MERS-s improves the sensitivity by

1033% over random test patterns [24], its best sensitivity is still too small, which is less than 1% in the majority benchmarks [24]. The low side-channel sensitivity in Reference [24] is due to the inherent restriction of reordering within the set of test patterns generated by MERS-s, which will be discussed in Section 5.4. MERS-s inherits the same bad performance of the N-detect approach and makes the test generation time even longer by reordering tests. The reordering step of MERS-s in maximizing side-channel sensitivity requires a large number of simulations and is not able to run in parallel (the same problem as the N-detect approach). Our proposed approach is able to effectively search for efficient tests that can drastically improve the side-channel sensitivity, and allow the searching process run in parallel—making Trojan detection feasible for large designs in practice.

2.3 Genetic Algorithm

Test generation using machine learning and simulated annealing methods have been extensively explored [25]. Genetic algorithm (GA) is a commonly used evolutionary search algorithm inspired by natural selection [36]. In the test generation domain, genetic algorithm is shown to be successful in fault coverage [45] and Trojan detection [46]. To the best of our knowledge, our work is the first attempt in utilizing genetic algorithm for side-channel analysis aware test generation.

3 PROBLEM FORMULATION AND MOTIVATION

3.1 Problem Formulation

In this work, we assume that the attackers are more likely to use rare nodes to construct hardware Trojans. Therefore, we use suspicious nodes and rare nodes interchangeably. The suspicious region is defined as the region that is less likely to be activated during traditional validation methodology using millions of random/constrained-random test patterns. Note that a design may have one or more suspicious regions. Moreover, each suspicious region can have one or more rare nodes with values below the rareness threshold.

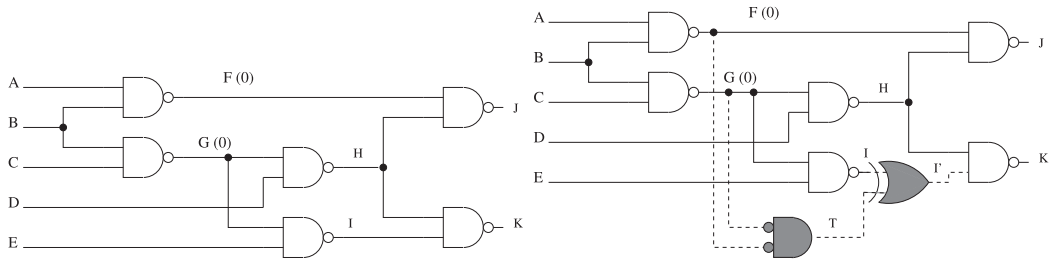
Our goal is to generate l compact ordered pairs of test patterns $\{<u_i,v_i>\}$ $(i = 1, 2, \ldots, l)$ that can maximize the dynamic current based side-channel sensitivity. For each pair of the test patterns $<u_i,v_i>$, the current switching in the golden design G (called the original switching) is measured by applying the first pattern u_i followed by the second pattern v_i, denoted $switch^G_{u_i,v_i}$. The current switching in the Trojan-inserted design G^T is defined in the same way, i.e., $switch^{G^T}_{u_i,v_i}$. The *relative switching* is computed as $|switch^G_{u_i,v_i} - switch^{G^T}_{u_i,v_i}|/switch^G_{u_i,v_i}$. The ***sensitivity*** of a Trojan T is defined as the maximum of the relative switching over all test patterns, as shown in Equation (1):

$$sensitivity_T = max_{(i=1,2,\ldots,l)}\left(\frac{|switch^G_{u_i,v_i} - switch^{G^T}_{u_i,v_i}|}{switch^G_{u_i,v_i}}\right). \quad (1)$$

3.2 An Illustrative Example

To illustrate how to improve the sensitivity in dynamic current-based side-channel analysis, we first use a small benchmark c17 from ISCAS-85 [1] as an example with its netlist shown in Figure 3(a). We set *rareness threshold* to be 0.3, i.e., rare nodes are defined as the signals whose rare values are satisfied with less than 30% probability in random simulations. For example, F and G are two rare nodes with rare value 0 in Figure 3(a).

Assume an attacker uses rare nodes F and G to construct the trigger condition, and the Trojan is shown using dashed lines in Figure 3(b). For this small design, we enumerate all possible pairs of test patterns and compute each sensitivity. The best pair of test patterns is $<u,v>$ = <11100,10100> on inputs (<A,B,C,D,E>), with only B switches from "1" to "0." The current switching in the golden



(a) The netlist of c17. (b) The netlist of c17 with a Trojan inserted.

Fig. 3. The netlist of c17 from ISCAS-85 [1] benchmark. We assume 0.3 as the threshold for rare nodes, F and G are the two rare nodes in this design, with their rare values 0. The top shows the golden design and the bottom shows the design with a Trojan inserted.

design $switch^{G}_{u,v}$ is 4 (switching of signals B, F, G, and J) and the current switching in the Trojan inserted design $switch^{G^t}_{u,v}$ is 7 (switching of signals B, F, G, J, T, I', and K). Thus, the sensitivity is 75% in this example. *An important observation is that by flipping only a small number of relevant inputs (B in this example) while preserving the others, the switching activities in the Trojan area are maximized while the current switching in the golden design is minimized.* In other words, if we can exploit the ***affinity*** between inputs and the rare nodes while creating a pair of test patterns (u followed by v), it can lead to a significant improvement in sensitivity for Trojan detection. Experimental results in Section 5.4 (Figure 9) demonstrate that affinity is useful in practice.

3.3 Motivation and Research Challenges

By inspecting the capability of <11100,10100> for c17, we want to divide the task of searching for effective pairs of test patterns into two sub-problems. (1) Generation of *the first pattern that tries to maximize activation of rare nodes with their respective rare values*, e.g., 11100 in the previous example. As the difference of current switching in designs with/without Trojans comes from the switching of the inserted circuits, the sensitivity can be improved if the switching activity is maximized in these suspicious regions. (2) Given the first pattern u generated in the previous step, searching for the most profitable *second pattern v, which is responsible for both maximization of switching in rare nodes and minimization of switching in non-rare nodes*, e.g., 10100 in the previous example.

However, there are three main challenges in searching for effective pairs of test patterns.

(1) Randomly selected pairs may not lead to high sensitivity, even if the two patterns are similar. For example, if we apply <u,v> = <11100,10100> to the previous example, the switching activities in G and G^T are the same, revealing no side-channel footprint.
(2) The whole search space is exponentially large (2^n, where n is the number of inputs in the design). So, searching for the whole space is not feasible. Based on affinity heuristic, the neighbor of u within a small Hamming distance (e.g., less than k) is the optimized search space. One naive way is to use breadth-first-search (BFS) according to the Hamming distance. However, the searching complexity is still $O(n^k)$.
(3) There is a tradeoff between introducing switching in the rare nodes and minimizing switching in the golden design. We need to introduce a reasonably large switching in rare nodes, since we have no knowledge of the trigger condition. However, for a design with thousands of rare nodes, introducing switching for all of them can lead to a significant increase in the switching of the golden design. In that case, even if the Trojan is fully

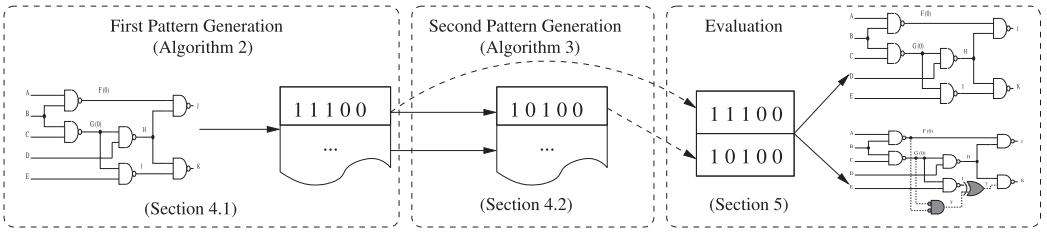


Fig. 4. The overview of our approach (MaxSense). We divide the task of test generation into two subproblems: (i) generation of the first patterns to maximize the activation of the rare nodes with their respective rare values; (ii) given the first pattern generated in the previous step, searching for the most profitable second pattern, which is responsible for both maximization of switching in the rare nodes and minimization of switching in the non-rare nodes.

activated, the sensitivity (tens of the extra switching divided by thousands of the original switching) can be too small compared to the process and noise margins.

Our approach addresses these challenges by using an SMT-based first pattern generation to maximize the activation of rare nodes and using genetic algorithm as an approximate and optimized replacement of BFS to search for the most profitable second patterns. Based on input affinity, we initialize GA with random test patterns that have fixed small Hamming distance from the first pattern. By crossover and mutation, the Hamming distance is expected to grow slowly. After several generations, the majority of the profitable test patterns in the expected search space are likely to be visited.

4 GENERATION OF EFFECTIVE TEST PATTERNS

Figure 4 shows an overview of our proposed approach (MaxSense). It has three important steps. The first step generates the first patterns to maximize activation of rare nodes with their respective rare values (Section 4.1). The next step finds the most profitable second patterns for both maximization of switching in rare nodes and minimization of switching in non-rare nodes (Section 4.2). Finally, we evaluate the quality of the generated pairs of test patterns (Section 5).

4.1 Generation of the First Patterns

The sensitivity of side-channel analysis is maximized if the ordered pairs of test patterns are able to maximize activation of rare nodes with their respective rare values, i.e., partially or fully activate trigger conditions. The basic idea is to increase the activities of rare signals to increase the probability of activating the unknown trigger conditions.

As discussed in Section 2.1, the test patterns generated by N-detect approach [14] is promising to achieve this goal. However, efficiency is the main bottleneck of the N-detect approach. The N-detect approach requires one simulation in each bit flipping of a single initial random test pattern. Therefore, if the number of initial random test vectors are large and the design is complex, it takes a long time to finish all simulations. What is worse, the N-detect approach cannot run in parallel. It is due to the fact that the N-detect criterion relies on the overall performance of all test vectors, and we cannot evaluate the quality of a single test pattern without evaluating all the other patterns. For example, if a set of test patterns already cover all but one rare signal N times, a new test pattern that is able to cover the remaining rare signal is better than a test vector that is able to cover hundreds of rare signals that are already activated by N times. To address these inherent problems of N−detect approach, we propose an effective and parallelizable test generation approach utilizing an SMT solver to produce the first patterns.

Before describing the details of our SMT-based approach, we first introduce logic expressions for signals that will be used in our approach. For each signal, we define its logic expression as an expression that is composed by controllable signals, such as primary inputs, flip-flops with a scan chain. For example, the logic expression for signal F is $!(A \wedge B)$ in Figure 3. This logic expression is stored in $F.expr$ and the rare value of F is stored in $F.rv$.

Algorithm 2 shows our SMT-based approach. The main part of Algorithm 2 contains M iterations, each of which generates one test pattern by calling an SMT solver. In each iteration, we first generate a random permutation of all rare signals in RP and initialize an SMT expression S to be true. Then, we construct an expression $e_i = (rp_i.expr == rp_i.rv)$ for each rare signal $rp_i \in RP$. These expressions from rare signals in RP are added one by one to our SMT expression S. To maintain S to be satisfiable, we skip every rare signal rp_i when $S \wedge e_i$ is unsatisfiable. In other words, S contains rare signals that can construct a valid trigger condition. When we have "enough" ($TriggerLimit$) satisfiable rare signals in S, we get an input pattern by solving S using an SMT solver. By choosing RP as a random permutation of all rare signals in each iteration, Algorithm 2 tries to generate test patterns that can activate different combinations of rare signals. At the end of Algorithm 2, M test patterns are returned as the set of first patterns, each of which is able to activate a combination of rare signals.

ALGORITHM 2: First Pattern Generation using SMT Solver

```
1:  procedure SMT(circuit netlist, rare signals RS, the number of test vectors M)
2:      initialize first pattern set FP = ∅
3:      compute logic expression for each rare signal
4:      for k = 1 to M do
5:          RP = random_permutation(RS)
6:          initialize SMT expression S = 1
7:          the total number of satisfiable expression total = 0
8:          for each rare signal rp_i ∈ RP do
9:              new expression e_i = (rp_i.expr == rp_i.rv)
10:             if satisfiable(S ∧ e_i) then
11:                 S = S ∧ e_i
12:                 total = total + 1
13:             end if
14:             if total > TriggerLimit then
15:                 break
16:             end if
17:         end for
18:         solve S and get input pattern u_k
19:         FP = FP ∪ {u_k}
20:     end for
21:     return FP
22: end procedure
```

The performance of Algorithm 2 depends on the total number of test patterns and the complexity of the design. The number of iterations is controlled by the user-defined parameter M, which is the size of test patterns. In each iteration, one random permutation of rare signals RS is performed with time complexity $O(|RS|)$. In the inner loop (Lines 8–17), we are looking for no more than $TriggerLimit$ rare signals that can be satisfied together. Therefore, the number of iterations of the inner loop is between $TriggerLimit$ and $|RS|$. In the worst case, all rare signals are tried and cannot find more than $TriggerLimit$ satisfiable rare signals. Therefore, the worst-case run time of Algorithm 2 is $O(M \times |RS| \times T(SMT))$, where $T(SMT)$ is the worst time to solve at most $TriggerLimit$

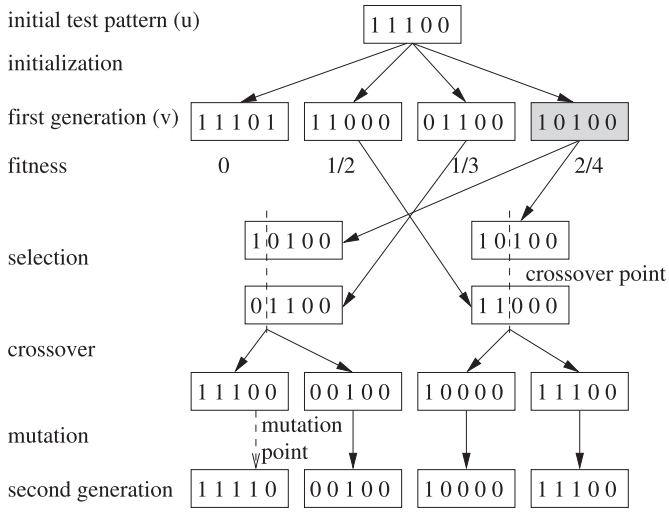


Fig. 5. The first iteration of GA for generating the best second pattern for $u = 11100$. The initial Hamming distance is 1 and the number of individuals for each generation is 4. Crossover point and mutation point are selected randomly. Fitness values are shown as the numerator representing rare switching $rare\_switch^G_{u_i,v}$ and the denominator representing total switching $switch^G_{u_i,v}$. The best second pattern v_i is the individual with the greatest fitness value over all generations as shown in the gray box.

expressions by an SMT solver. The running time can be reduced with multi-core architectures. It is easy to see that Lines 5–18 can be run in parallel. In other words, if we want to generate M test vectors and we have C cores, each core can generate M/C test vectors, i.e., C times speedup.

4.2 Searching for the Most Profitable Second Patterns

The second task is to find the best second pattern v_i for each u_i (identified in Section 4.1), such that the relative switching is maximized. There are many selection algorithms in the literature, including genetic algorithm, simulated annealing, and machine learning. While all of them provided promising results, we used genetic algorithm in our framework primarily due to its effectiveness in exploiting affinity and delivering profitable second patterns in a small number of iterations, as described in Section 5.4.

Genetic algorithm forms the main part of Algorithm 3, which consists of four major steps: initialization, fitness computation, selection, and crossover and mutation. The ***fitness*** is defined in Equation (2), where $rare\_switch^G_{u,v}$ represents the current switching of all rare nodes in G when applying the test pattern u followed by v. A profitable test pattern should maximize the current switching in rare nodes to increase the probability of activating a Trojan and minimize the total switching in the golden design. The best second pattern v_i for a given preceding u_i is the one achieving the highest fitness value over all generations (line 11). The first iteration of GA for c17 is shown in Figure 5, assuming 4 individuals in each generation.

$$fitness_u(v) = \frac{rare\_switch^G_{u,v}}{switch^G_{u,v}}. \tag{2}$$

4.2.1 Initialization. The first population is initialized with random test patterns that are similar to u_i. Each individual in the initial population has a Hamming distance k from u_i. During the experiments, we choose k to be $\max(0.004|u_i|, 1)$. Starting from a very low Hamming distance

ALGORITHM 3: Second Pattern Generation using GA

```
1: procedure GA(circuit netlist, rare signals RS, first patterns FP = {u_i})
2:     for each first pattern u_i ∈ FP do
3:         Initialization of GA with u_i
4:         For each individual v, compute fitness_{u_i}(v) by simulating the netlist with the pair of test patterns <u_i, v>
5:         for gen = 1 to generations do
6:             Selection of parents from the gen^th generation based on fitness values
7:             Single point crossover to produce children
8:             Single point mutation according to mutation rate
9:             Compute fitness for the children ((gen + 1)^th generation)
10:        end for
11:        Select the best individual over all generations as v_i
12:    end for
13:    return ordered pairs of test patterns {<u_i,v_i>}
14: end procedure
```

may potentially give us the benefit of a small original switching in the golden model, which may lead to a large fitness value based on Equation (2).

4.2.2 Fitness Computation. For each individual v, the golden design G is simulated with the pair of test patterns $<u_i,v>$. Then the fitness of v is computed by Equation (2). For example, the fitness values for four candidates are shown in Figure 5.

4.2.3 Selection. Selection is based on the fitness of each individual. An individual with a greater fitness is more likely to be selected. The selection shown in Figure 5 demonstrates that the individual with a greater fitness (such as 10100) is more likely to be selected than the one with a smaller fitness (such as 11101).

4.2.4 Crossover and Mutation. A single crossover point is randomly selected and crossover is performed on parents to produce two children. During mutation, a randomly selected position is mutated with a low mutation rate. For example, Figure 5 shows only 1 mutation for 4 individuals.

Although the Hamming distance to u_i is small in all individuals of the initial generation, crossover and mutation will increase the Hamming distance from generation to generation. Theoretically, the largest possible Hamming distance between the jth generation and u_i is at most $2^j * |k|$ considering only crossover. For all test patterns to be evaluated with some probability, the total number of generations should be large enough to allow $|u_i|$ Hamming distance. However, we may need only a small number of generations, as the affinity heuristic suggests. During experiments, we fix the number of generations to be 5. So, the maximum Hamming distance could be $2^5 \times \max(0.004|u_i|, 1)$, which is around $\max(10\%|u_i|, 32)$. Based on the affinity observation discussed in Section 3.2, it is likely to find a high-quality test pair within this search space. As demonstrated in Section 5.4 (Figure 9), the pairs of test patterns with small Hamming distances are effective in providing a significant improvement in sensitivity.

Since lines 3–11 are performing an independent GA searching for each first pattern u_i, the body of the first for-loop can run in parallel similar to Algorithm 2. Therefore, C cores will give a speedup close to C. Combined with Algorithm 2 to generate first patterns, the whole process can run in parallel. Given unlimited computing resources, we can use each core to generate one pair and combine all pairs together. This optimistic overall running time consists of compiling the design, generating one first pattern from Algorithm 2 and its corresponding second pattern from Algorithm 3.

4.3 Selection of *TriggerLimit*

As introduced in Section 3.3, there is a tradeoff between introducing switching in the rare nodes and minimizing switching in the golden design. We try to address this challenge by selecting a reasonable *TriggerLimit*. This section illustrates why it is a challenge to select a reasonable *TriggerLimit*—a larger *TriggerLimit* can lead to increased total switching (and reduced sensitivity), while a smaller *TriggerLimit* can lead to reduced probability of Trojan activation.

TriggerLimit in Algorithm 2 controls how many rare signals should be activated by each first pattern u. Its second pattern generated by Algorithm 3 tends to introduce as many switching in the rare nodes activated by u as possible to increase fitness value. First, we show that if the *TriggerLimit* is too large, it may lead to sub-optimal patterns with lower sensitivity. Assume that the inserted Trojan has 8 trigger points and we compare $TriggerLimit = 128$ to $TriggerLimit = 8$. (i) With $TriggerLimit = 128$, suppose that we generate a first pattern u' that is able to activate 128 rare nodes (include all 8 trigger points of the Trojan) and find a second pattern v' by Algorithm 3 that introduces current switching in all of these 128 rare nodes with a total switching of 1,000. (ii) With $TriggerLimit = 8$, suppose that we get a first pattern u^* that is able to activate only the 8 trigger points of the Trojan and find a second pattern v^* that introduces current switching in all of these 8 rare nodes with a total switching of 500. When these two test vectors ($<u',v'>$ versus $<u^*,v^*>$) are applied to the Trojan inserted design, the switching from the Trojan will be same, but the sensitivity produced by $<u^*,v^*>$ will be greater than $<u',v'>$, since the original switching produced by $<u^*,v^*>$ is smaller. Next, we show that if *TriggerLimit* is too small, it may be not beneficial for Trojan detection. In the above example, the probability of the 128 rare nodes activated by u' containing all 8 trigger points of the Trojan is high, while the probability of the 8 rare nodes activated by u^* being the exact 8 rare triggers points of the Trojan is very low. It is obvious that if we introduce less switching in the Trojan area, the sensitivity will be smaller.

In summary, with a small *TriggerLimit*, e.g., 8 in the previous example, the probability of activating unknown Trojans is extremely low compared to $TriggerLimit = 128$ with the same number of test patterns. With a large *TriggerLimit*, e.g., 128 in the previous example, Algorithm 3 will try to maximize the fitness value by introducing as many switching in the rare nodes as possible, and as a result, introducing a large original switching and low sensitivity in Equation (1). In practice, we set *TriggerLimit* to be a few times of the largest possible trigger points. *TriggerLimit* is typically small, since the number of trigger points is small; otherwise, it will introduce noticeable power or area overhead, which is easier for debug engineer to detect using side-channel analysis. In our experiments, *TriggerLimit* is set to be 32 for Trojans with 8 trigger points.

5 EXPERIMENTS

To evaluate the effectiveness of our approach, we did a variety of experiments to show the results on different benchmarks and compared the results to the state-of-the-art approach. In addition, we also evaluated the efficiency of our approach utilizing multi-core systems. Note that although we evaluate the effectiveness of the generated test patterns using gate-level simulation, the test patterns generated by our approach are also applicable on post-silicon designs (fabricated chips).

5.1 Experimental Setup

All algorithms of our framework are implemented in C++, and Algorithm 2 utilizes Z3 [19] C++ as our SMT solver. Since MERS [24] is the state-of-the-art (closest to our approach), we used the same benchmarks as MERS—a subset of ISCAS-85 [1] and ISCAS-89 [2] gate-level benchmark circuits. We have also used two large benchmarks, memory controller (MC) from TrustHub [4] and MIPS processor from OpenCores [3], to demonstrate the scalability of our approach. We performed a

Table 1. Comparison of MaxSense with NDT+GA [30] and MERS-s [24] over 1,000 Random 8-trigger Trojans

Bench	#rare signals	MERS-s [24]			NDT + GA [30]				MaxSense				
		Sensitivity	% of Trojans detected	Time (hour)	Sensitivity	Imp. over [24]	% of Trojans detected	Time (hour)	Sensitivity	Imp. over [24]	% of Trojans detected	Time (hour)	Imp. over [24]
c2670	43	5.6%	2.4%	5.1	110%	19.6×	100%	1.1	133%	23.8×	100%	0.3	4.6×
c5315	164	1.5%	0.3%	23.5	52.4%	34.9×	98.3%	4.1	163%	108.7×	99.4%	0.9	26.1×
c7552	278	1.7%	0.2%	40.3	34.5%	20.3×	82%	7.9	51.9%	30.5×	83.5%	1.4	5.1×
s13207	604	1.7%	0%	11.5	79.7%	46.9×	100%	12.9	99%	58.2×	100%	0.7	16.4×
s15850	649	1.0%	0%	16.1	54.3%	54.3×	99.5%	15.1	55.7%	55.7×	97.1%	0.9	17.9×
s35932	1,152	1.0%	0.1%	10.7	52.9%	52.9×	66.6%	29.6	95.7%	95.7×	99.8%	1.8	5.9×
MC	1,306	-	-	-	-	-	-	-	14.9%	∞	85.2%	5.7	∞
MIPS	906	-	-	-	-	-	-	-	35.1%	∞	55.2%	13.0	∞
Avg.[1]	-	2%	0.5%	17.9	64.0%	38.2×	91.1%	11.8	99.7%	62.1×	96.6%	1	12.7×

MaxSense can provide an order-of-magnitude improvement in sensitivity. As a result, MaxSense can detect almost all the Trojans in majority of the benchmarks, while MERS-s fails to detect almost all of them.
[1]Since MERS-s [24] and NDT+GA [30] cannot finish in the MC and MIPS, all average results are computed over ISCAS benchmarks for comparison.

variety of experiments on a machine with Intel Xeon E5-2698 CPU @2.20 GHz. We compared three approaches with the configurations shown below:

(1) **MaxSense** (Our approach): MaxSense utilizes SMT (Algorithm 2) to generate the first patterns and genetic algorithm (Algorithm 3) to generate the second patterns. We fixed the number of test patterns M = 5,000 for ISCAS benchmarks, and M = 10,000 for MC and MIPS in Algorithm 2. In Algorithm 3, we set the number of individuals to be 200 in each generation, the number of generations to be 5, and mutation rate to be 0.1.
(2) **NDT+GA** [30]: It is our earlier work (conference version [30]) to show GA is more suitable than reordering for sensitivity improvement. It utilized N-detect [14] to generate the first patterns and genetic algorithm (Algorithm 3, the same configuration as GA in MaxSense) to generate the second patterns. We fixed N=1,000 for the N-detect criterion. To increase the quality of the generated tests, we set different parameters in this experimental setting compared to Reference [30]. Algorithm 1 starts with 100,000 random patterns for ISCAS benchmarks (compared to 10,000 random patterns in Reference [30]) and 1M random patterns for MC and MIPS. It also increases the number of rounds for bit flipping to two (compared to one in Reference [30]) for every initial random test pattern.
(3) **MERS-s** [24]: It is the state-of-the-art approach from Reference [24] (MERS with simulation-based reordering) with the best settings (C = 5.0) [24]. We did not compare with random tests and MERS (with Hamming distance), since MERS-s outperforms them.

5.2 Generation of Hardware Trojans

To statistically evaluate the performance of different approaches, we first generated 1,000 valid Trojans for each benchmark. To make these Trojans covert under traditional validation, we constructed each Trojan with a number of rare signals (trigger points) defined by a rareness threshold for each benchmark. To get the list of all rare signals, we first ran 1M random simulations for each benchmark. We set rareness threshold to be 0.1 for ISCAS benchmarks and 0.005 for MC and MIPS over all experiments. The number of rare signals is shown in the second column of Table 1. As we can see, the number of rare signals are around 1,000 for large benchmarks. After achieving the rare

signal list, different combinations of rare signals are randomly sampled and fed into ATPG tool to check their validity. Finally, 1,000 valid Trojans are independently inserted into the benchmarks to construct design under tests (DUTs). These constructed 1,000 DUTs for each benchmark are used to evaluate the performance of different approaches over all the experiments. It is easy to see that the probability of activating these Trojans using random simulation is at most 10^{-p} for ISCAS benchmarks, and 200^{-p} for large benchmarks, assuming each Trojan is constructed by p independent rare signals. In all of our experiments, the trigger points p is set to be 8. The possible combinations that an adversary can exploit to insert Trojans are in the order of $\binom{1000}{p}$ with 1,000 rare signals, which is around 10^{19} for $p = 8$.

5.3 Performance Evaluation

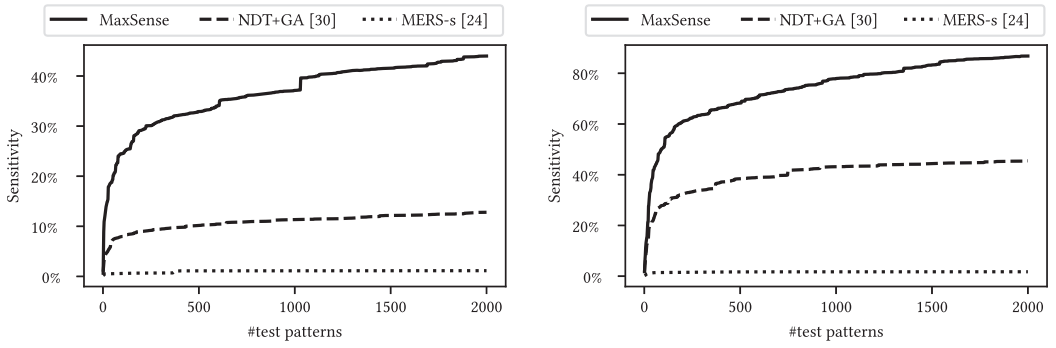
We applied the test patterns generated by the three approaches in Section 5.1 to simulate both the golden design and all DUTs in Section 5.2. We computed the side-channel sensitivity in current switching according to Equation (1). For each DUT, we conclude the existence of a Trojan if the sensitivity is greater than 10% [9].

The results are shown in Table 1. For each approach, we report the sensitivity, the percentage of detected Trojans, and the overall running time. The sensitivity is the average sensitivity over 1,000 randomly sampled Trojans, each of which is computed using Equation (1) with all the test patterns. The percentage of detected Trojans shows the fraction of Trojans whose sensitivities are above the threshold 10%. The running time of MERS-s consists of the generation of MERS test patterns and simulation-based reordering. The running time of NDT+GA and MaxSense consist of the running time of N-detect [14] and SMT (Algorithm 2), respectively, and the running time of genetic algorithm (Algorithm 3). The entry marked with dash represents test generation timeout after one week.

5.3.1 Sensitivity Comparison. The sensitivity is the most important indicator of the quality of the generated test patterns, since a larger sensitivity can help detect more Trojans directly or by sophisticated classification approaches. As shown in Table 1, the overall sensitivity of MaxSense outperforms MERS-s by 62 times, and the NDT+GA improves a factor of 38 over MERS-s. For the ISCAS benchmarks, while the test patterns generated by MERS-s only get less than 2% sensitivity except for the smallest one c2670, NDT+GA and MaxSense improve the sensitivity to 64% and 99.7% on average, respectively. For the two large benchmarks, MERS-s and NDT+GA cannot finish within one week time limit due to the long running time of N-detect approach in these two approaches and also reordering in MERS-s. However, MaxSense is able to provide 14.9% and 35.1% sensitivity in MC and MIPS, respectively, with a few hours of test generation time.

By comparing the performance of NDT+GA and MaxSense, we can observe that SMT-based approach achieves 63% higher sensitivity compared to N-detect based approach. It shows that the first patterns generated by SMT are better than N-detect in activating rare nodes. While N-detect approach increases the activation of rare nodes by flipping bits from random test patterns, SMT-based approach directly controls which parts of rare nodes to be activated. Therefore, it is expected that SMT-based approach would outperform N-detect-based approach. Compared to reordering technique in MERS-s, our genetic algorithm generates the most profitable second patterns for a given first pattern, leading to a significant reduction in the original switching, which will be explained in Section 5.4.

To inspect the effectiveness of one test pattern pair, we compute the average sensitivity of all DUTs from c7552 and s13207 after applying the first 2,000 test pattern pairs in Figure 6. Both Figure 6(a) and Figure 6(b) reveal the same pattern that the average sensitivity grows significantly faster by the test patterns from MaxSense than NDT+GA and MERS-s. When the number of test



(a) c7552 (b) s13207

Fig. 6. The average sensitivity of 1,000 DUTs after applying the first 2,000 pairs of test patterns. The average sensitivity by MaxSense grows fastest and achieves the best performance. NDT+GA grows fast at the beginning and soon becomes saturated. MERS-s performs the worst with less than 2% sensitivity for both benchmarks.

patterns is below 500, MaxSense is already able to generate more than 30% and 60% average sensitivity in c7552 and s13207, respectively, while NDT+GA achieves 10% and 40% in these two benchmarks. The average sensitivity of NDT+GA grows fast in the first few hundreds of test patterns, but it saturates soon. MERS-s performs the worst, with less than 2% in both benchmarks after 2,000 test patterns. Therefore, the test patterns generated by MaxSense is more effective and more compact than NDT+GA and MERS-s. Among these three approaches, the test patterns generated by MERS-s have the worst quality that achieve an average sensitivity typically less than 2%, which is far less than process variation and environment noise.

5.3.2 Detected Trojans. With the assumption of 10% sensitivity threshold [9], the percentage of detected Trojans are shown in Table 1. Since the sensitivity from MERS-s are mostly less than 2%, MERS-s missed almost all the Trojans. However, NDT+GA and MaxSense are able to detect 91.1% and 96.6%, respectively, of Trojans on ISCAS benchmarks.

The cumulative distributions of the sensitivities over 1,000 Trojans in c7552, s13207, s15850, and MIPS are shown in Figure 7. The x-axis represents the sensitivity, y-axis represents the number of Trojans that have sensitivities greater than x, and the vertical line represents 10% sensitivity threshold (noise). For example, in s13207, almost all the Trojans have sensitivities greater than the sensitivity threshold in both MaxSense and NDT+GA, while in MERS-s this number is 0. In other words, if we assume the process variation to be 10%, MaxSense and NDT+GA can detect the majority of these randomly sampled Trojans with high confidence, while MERS-s missed almost all of them. The exact numbers are reported in Table 1. When the sensitivity threshold increases (the vertical bar moves to the left), the number of detected Trojans drops. It is expected, since when the noise increases, the differences of current switching in some DUTs are not significant enough to conclude the existence of hardware Trojans. As shown in Figure 7, when the noise increases from 10% to 25% (the vertical bar moves to 2^{-2}), the detection rates of c7552 drop to 60% and 50% for MaxSense and NDT+GA, respectively. Overall, our approach can detect majority of the Trojans in most of the benchmarks due to the higher sensitivities provided by our test patterns, whereas the test patterns generated by MERS-s can barely detect any Trojans in a practical noisy settings.

5.3.3 Test Generation Time. Finally, we compare the test generation time of the three approaches. As shown in Table 1, while MERS-s and NDT+GA requires 17.9 and 11.8 hours on

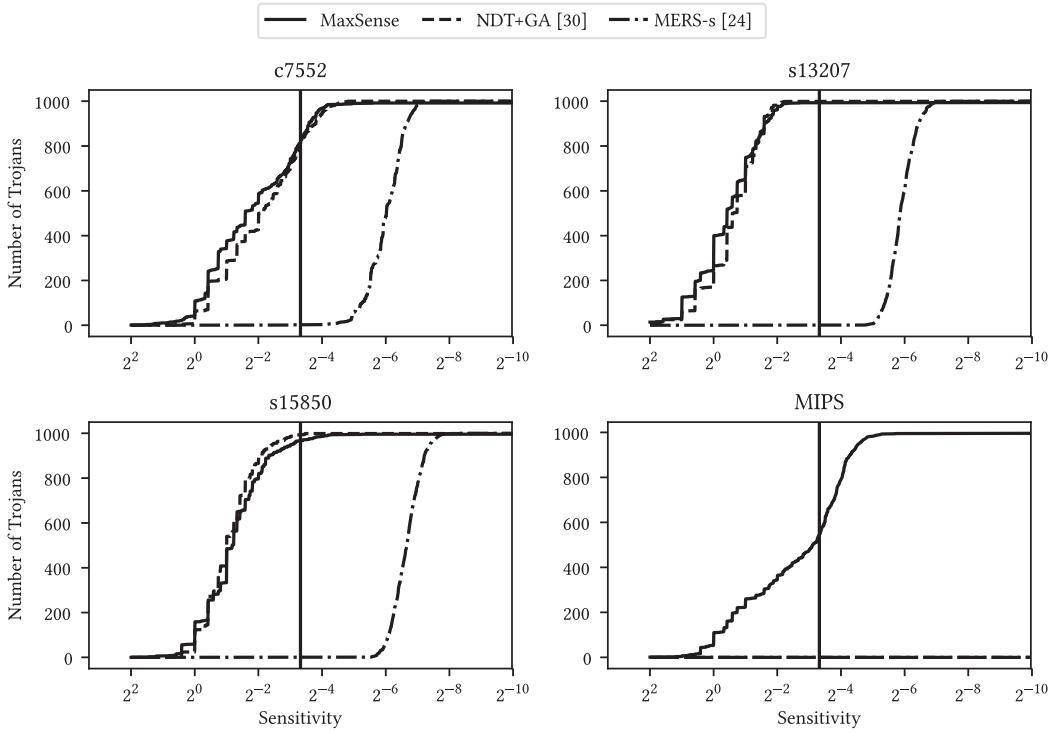
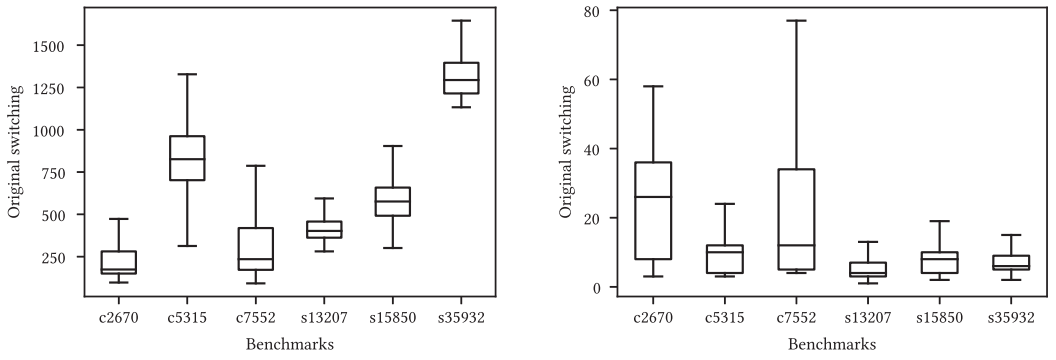


Fig. 7. The comparison of distributions of sensitivities by three approaches over 1,000 Trojans. The x-axis shows the sensitivity in logarithmic axis, and y-axis is the number of Trojans that have sensitivities greater than x. The vertical line represents 10% process variations as a threshold to detect Trojans.

average, respectively, to generate test patterns for ISCAS benchmarks, MaxSense takes only 1 hour. MaxSense is up to 26.1×, 12.7× on average, more efficient than MERS-s. For the two large benchmarks, MC and MIPS, MERS-s cannot finish in one week, while MaxSense can generate high quality test in several hours. It indicates that the improvement increases when the size of design becomes larger and larger. Furthermore, MaxSense is able to utilize multi-core platforms to reduce the running time by several times for large designs (see Section 5.6), while MERS-s is not suitable for even medium-size designs. The long running time of MERS-s is due to the usage of N-detect approach in generating test patterns, and time-consuming simulations in reordering test patterns. The reordering in MERS-s takes $O(n^2)$ simulations for each pair of test patterns, where n is the number of test patterns. However, the number of simulations in our genetic algorithm is $O(n)$, where n is the number of first test patterns, since we evaluate 5 generations and 200 individuals for each generation. Moreover, the test patterns generated by MaxSense is more compact than MERS-s, as shown in Figure 6. Therefore, the linear growth in the number of simulations in Algorithm 3 leads to significantly faster test generation than the quadratic growth in the number of simulations in the reordering part of MERS-s.

5.4 Evaluation of Original Switching

Because the inserted Trojans are usually tiny compared to the whole design and the probability of fully activating a Trojan is low, it is critical to minimize the current switching in the original design such that the sensitivity is high enough to be detected. Figure 8 shows the box plot of all



(a) MERS-s [24] (b) MaxSense

Fig. 8. The distribution of the original switching (denominator of sensitivity in Equation (1)) when applying all test patterns to the golden design. Large original switching is the main reason of low sensitivity in MERS-s.

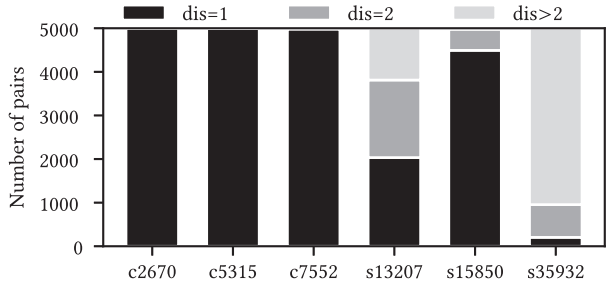
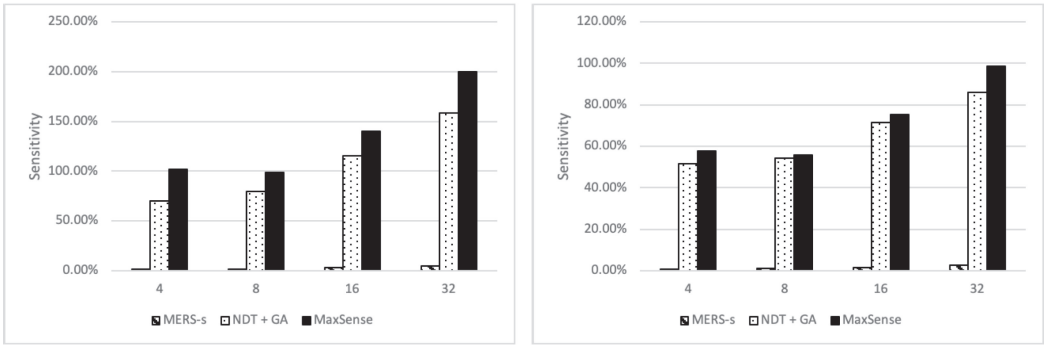


Fig. 9. Hamming distance of all pairs of test patterns by MaxSense.

original switching from test patterns generated by MERS-s and MaxSense for ISCAS benchmarks. Figure 8(a) shows that the original switching from MERS-s is in the order of several hundreds, up to 1,500. Figure 8(b) shows that MaxSense achieves less than 100 original switching for all benchmarks, with the average original switching around 10 to 20. Compared to MERS-s, MaxSense is able to achieve up to more than 100 times reduction in the original switching, e.g., the median of original switching in s35932 is around 1,250 from MERS-s and less than 10 from MaxSense. With an 8-trigger Trojan, the number of extra current switching is typically less than 16, assuming the Trojan is not fully activated. Therefore, the large original switching becomes the main bottleneck that prevents MERS-s from achieving a high sensitivity.

The large original switching from MERS-s is due to its reordering technique. As the reordering of MERS-s restricts each test pattern to find its pair from the generated test patterns, the minimum original switching that reordering can achieve is bounded by the optimum pairs inside these test patterns. However, MaxSense fixes the first pattern and searches an open space for profitable second patterns to minimize the original switching. The searching process starts with test patterns that are close to the first pattern and gradually increases the distance using genetic algorithm. For each first test pattern u from SMT (Algorithm 2), the best second pattern v is found by GA with 5 generations. Thus, the maximum possible Hamming distance between u and v can be as large as $10\%|u|$ (see Section 4.2). It follows the motivation of affinity heuristic introduced in Section 3. We examined the Hamming distance between the generated pairs of MaxSense in Figure 9. It is interesting to see that almost all of the distances are 1 in the relatively small circuits from ISCAS-85 [1].



(a) s13207 (b) s15850

Fig. 10. The sensitivity with various trigger points.

They reveal a similar property as the example in Figure 3 that one bit change can maximize the activity in the Trojan area and minimize the activity in the remaining of the design. When the size of the circuit grows, the Hamming distance of the test pairs that provide the highest sensitivity also increases. For example, the majority of the distances for s35932 are greater than 2.

5.5 Evaluation of the Number of Trigger Points

In this experiment, we evaluated the effects of the trigger point number on the sensitivity. In theory, a hardware Trojan with more trigger points typically introduces more side-channel effects. To investigate the effects of the trigger points, we selected two large benchmarks that all of the three approaches can finish, s13207 and s15850. Then, we used the same method in Section 5.2 to generate 1,000 hardware Trojans with each trigger point number, which varies from 4 and 32. Finally, we applied the three approaches to both benchmarks and computed the average sensitivities. Figure 10 shows the average sensitivities with different number of trigger points. When the trigger points increase from 4 to 8, the average sensitivities do not change much. When the trigger points keep increasing, the sensitivities have a significant increase. While the sensitivities of MERS-s grow with the increasing number of trigger points, the sensitivities of MERS-s are still far from the noise threshold even with 32 trigger points.

5.6 Concurrency of MaxSense

As discussed in Section 2 and Section 4, N-detect approach cannot run in parallel, since the N-detect criterion relies on the overall performance of all test vectors. Although the second step of NDT+GA can run in parallel, it does not benefit much from multi-core platforms, since N-detect part consumes the majority of test generation time. Similarly, MERS-s cannot utilize multi-core platforms because of the N-detect criterion. What is worse, the reordering step of MERS-s cannot run concurrently either. As a result, only MaxSense is evaluated in this experiment, whose both steps can run in parallel.

To evaluate the performance of MaxSense in a multi-core platform, we tested the test generation time of MIPS with 1, 2, 4, 8, and 16 threads running in different cores. Multi-threading scheme is implemented using C++ pthread library. Each thread is responsible to generate its own pairs of test patterns. For example, to generate 10,000 pairs of test patterns in a 16-core platform, each thread is responsible to generate 625 pairs of test patterns using Algorithm 2 with $M = 625$ followed by Algorithm 3. The overhead of multi-threading includes compiling the design for individual simulation of each thread and returning the generated test pattern pairs to the main thread.

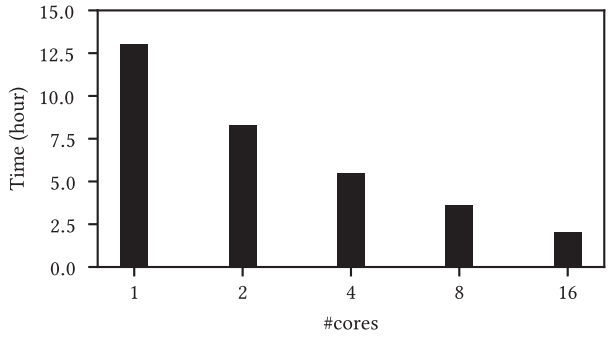


Fig. 11. The test generation time of MaxSense with multi-core platforms for MIPS processor. With 16 cores, MaxSense can finish generating 10,000 test pattern pairs in less than 2 hours.

The test generation time of MaxSense is shown in Figure 11. It is clear to see that MaxSense can achieve better performance in multi-core platforms, since both its pairs of test patterns can be generated in parallel. With 2 threads, MaxSense can achieve speedup around 1.6×. The final speedup using 16 cores is more than 6 times, leading to less than 2 hours in generating 10,000 test pattern pairs. It is significantly faster than MERS-s, which takes longer than one week to finish. With enough cores, the best possible performance would be the total time for compiling the design and generating one pair of test patterns. Since compilation time is the dominant factor and it is linearly related to the design size, the time complexity of MaxSense would be linear with respect to the design size with enough cores.

6 CONCLUSION

Side-channel analysis provides a promising approach for Trojan detection. The state-of-the-art test generation technique (e.g., MERS-s [24]) is not beneficial for large designs due to its high run-time complexity. Most importantly, the sensitivity obtained by the existing approaches is very low compared to environmental noise and process variations, making them useless in practice. Our proposed approach addresses both limitations by developing an SMT-based first pattern generation algorithm and a genetic algorithm-based second pattern generation algorithm that can increase the sensitivity drastically while significantly reduce the test generation time. Our approach breaks down the problem into two sub-problems. The first task generates effective test patterns to maximize the excitation of rare values. The second task finds the best matching pair for each test pattern generated in the first task to maximize the sensitivity. In this article, we demonstrated that the combination of the SMT-based approach with the genetic algorithm can generate significantly better test patterns than MERS-s. Our proposed test generation approach can improve both side-channel sensitivity (up to 109×, 62× on average) and test generation time (up 26×, 13× on average) compared to MERS-s. Experimental results demonstrated that our approach can detect the majority of Trojans in the presence of process variation and noise margins while the state-of-the-art approaches fail.

REFERENCES

[1] [n.d.]. ISCAS85 Combinational Benchmark Circuits. Retrieved on May 2020 from https://filebox.ece.vt.edu/~mhsiao/iscas85.html.
[2] [n.d.]. ISCAS89 Sequential Benchmark Circuits. Retrieved from https://filebox.ece.vt.edu/~mhsiao/iscas89.html.
[3] [n.d.]. OpenCores. Retrieved from https://www.opencores.org/.
[4] [n.d.]. TrustHub. Retrieved from https://www.trust-hub.org/.

[5] I. H. Abbassi, F. Khalid, O. Hasan, A. M. Kamboh, and M. Shafique. 2018. McSeVIC: A model checking based framework for security vulnerability analysis of integrated circuits. *IEEE Access* 6 (2018), 32240–32257. DOI: https://doi.org/10.1109/ACCESS.2018.2846583

[6] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar. 2007. Trojan detection using IC fingerprinting. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'07)*. 296–310. DOI: https://doi.org/10.1109/SP.2007.36

[7] Alif Ahmed, Farimah Farahmandi, Yousef Iskander, and Prabhat Mishra. 2018. Scalable hardware Trojan activation by interleaving concrete simulation and symbolic execution. In *Proceedings of the IEEE International Test Conference (ITC'18)*.

[8] M. E. Amyeen, S. Venkataraman, A. Ojha, and Sangbong Lee. 2004. Evaluation of the quality of N-detect scan ATPG patterns on a processor. In *Proceedings of the International Conference on Test*. 669–678.

[9] Bharathan Balaji, John McCullough, Rajesh K. Gupta, and Yuvraj Agarwal. 2012. Accurate characterization of the variability in power consumption in modern mobile processors. In *Proceedings of the Workshop on Power-aware Computing and Systems*. USENIX. Retrieved from https://www.usenix.org/conference/hotpower12/workshop-program/presentation/Balaji.

[10] Mainak Banga, Maheshwar Chandrasekar, Lei Fang, and Michael S. Hsiao. 2008. Guided test generation for isolation and detection of embedded trojans in ICs. In *Proceedings of the 18th ACM Great Lakes Symposium on VLSI (GLSVLSI'08)*. ACM, New York, NY, 363–366. DOI: https://doi.org/10.1145/1366110.1366196

[11] C. Bao, D. Forte, and A. Srivastava. 2015. Temperature tracking: Toward robust run-time detection of hardware Trojans. *IEEE Trans. Comput.-aided Des. Integ. Circ. Syst.* 34, 10 (Oct. 2015), 1577–1585. DOI: https://doi.org/10.1109/TCAD.2015.2424929

[12] Amin Bazzazi, Mohammad Taghi Manzuri Shalmani, and Ali Mohammad Hemmatyar. 2017. Hardware Trojan detection based on logical testing. *J. Electron. Test.* 33, 4 (Aug. 2017), 381–395. DOI: https://doi.org/10.1007/s10836-017-5670-0

[13] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan. 2014. Hardware Trojan attacks: Threat analysis and countermeasures. *Proc. IEEE* 102, 8 (Aug. 2014), 1229–1247. DOI: https://doi.org/10.1109/JPROC.2014.2334493

[14] Rajat Subhra Chakraborty, Francis Wolff, Somnath Paul, Christos Papachristou, and Swarup Bhunia. 2009. *MERO: A Statistical Approach for Hardware Trojan Detection*. Springer Berlin, 396–410.

[15] Mingsong Chen, Xiaoke Qin, Heon-Mo Koo, and Prabhat Mishra. 2012. *System-level Validation: High-level Modeling and Directed Test Generation Techniques*. Springer Publishing Company, Incorporated.

[16] X. Chen, L. Wang, Y. Wang, Y. Liu, and H. Yang. 2017. A general framework for hardware Trojan detection in digital circuits by statistical learning algorithms. *IEEE Trans. Comput.-aided Des. Integ. Circ. Syst.* 36, 10 (Oct. 2017), 1633–1646. DOI: https://doi.org/10.1109/TCAD.2016.2638442

[17] F. Courbon, P. Loubet-Moundi, J. J. A. Fournier, and A. Tria. 2015. A high efficiency hardware Trojan detection technique based on fast SEM imaging. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE'15)*. 788–793. DOI: https://doi.org/10.7873/DATE.2015.1104

[18] J. Cruz, F. Farahmandi, A. Ahmed, and P. Mishra. 2018. Hardware Trojan detection using ATPG and model checking. In *Proceedings of the 31st International Conference on VLSI Design and the 17th International Conference on Embedded Systems (VLSID'18)*. 91–96. DOI: https://doi.org/10.1109/VLSID.2018.43

[19] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. 337–340.

[20] F. Farahmandi, Y. Huang, and P. Mishra. 2017. Trojan localization using symbolic algebra. In *Proceedings of the 22nd Asia and South Pacific Design Automation Conference (ASP-DAC'17)*. 591–597. DOI: https://doi.org/10.1109/ASPDAC.2017.7858388

[21] Farimah Farahmandi, Yuanwen Huang, and Prabhat Mishra. 2019. *System-on-Chip Security: Validation and Verification*. Springer Nature.

[22] S. Ghosh, A. Basak, and S. Bhunia. 2015. How secure are printed circuit boards against Trojan attacks? *IEEE Des. Test* 32, 2 (Apr. 2015), 7–16. DOI: https://doi.org/10.1109/MDAT.2014.2347918

[23] Yuanwen Huang, Swarup Bhunia, and Prabhat Mishra. 2016. MERS: Statistical test generation for side-channel analysis based Trojan detection. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*. ACM, New York, NY, 130–141. DOI: https://doi.org/10.1145/2976749.2978396

[24] Y. Huang, S. Bhunia, and P. Mishra. 2018. Scalable test generation for Trojan detection using side channel analysis. *IEEE Trans. Inf. Forens. Secur.* 13, 11 (Nov. 2018), 2746–2760. DOI: https://doi.org/10.1109/TIFS.2018.2833059

[25] Charalambos Ioannides and Kerstin I. Eder. 2012. Coverage-directed test generation automated by machine learning—A review. *ACM Trans. Des. Autom. Electron. Syst.* 17, 1 (Jan. 2012). DOI: https://doi.org/10.1145/2071356.2071363

[26] D. Ismari, J. Plusquellic, C. Lamech, S. Bhunia, and F. Saqib. 2016. On detecting delay anomalies introduced by hardware Trojans. In *Proceedings of the IEEE/ACM International Conference on Computer-aided Design (ICCAD'16)*. 1–7. DOI: https://doi.org/10.1145/2966986.2967061

[27] N. Jacob, D. Merli, J. Heyszl, and G. Sigl. 2014. Hardware Trojans: Current challenges and approaches. *IET Comput. Dig. Tech.* 8, 6 (2014), 264–273. DOI: https://doi.org/10.1049/iet-cdt.2014.0039

[28] Yier Jin and Y. Makris. 2008. Hardware Trojan detection using path delay fingerprint. In *Proceedings of the IEEE International Workshop on Hardware-oriented Security and Trust*. 51–57. DOI: https://doi.org/10.1109/HST.2008.4559049

[29] Yangdi Lyu and Prabhat Mishra. 2018. A survey of side-channel attacks on caches and countermeasures. *J. Hardw. Syst. Secur.* 2, 1 (01 Mar. 2018), 33–50. DOI: https://doi.org/10.1007/s41635-017-0025-y

[30] Yangdi Lyu and Prabhat Mishra. 2019. Efficient test generation for Trojan detection using side channel analysis. In *Proceedings of the Design Automation and Test in Europe Conference (DATE'19)*.

[31] Yangdi Lyu and Prabhat Mishra. 2020. Automated test generation for activation of assertions in RTL models. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC'20)*.

[32] Yangdi Lyu and Prabhat Mishra. 2020. Automated test generation for Trojan detection using delay-based side channel analysis. In *Proceedings of the Design Automation and Test in Europe Conference (DATE'20)*.

[33] Yangdi Lyu and Prabhat Mishra. 2020. Scalable activation of rare triggers in hardware Trojans by repeated maximal clique sampling. *IEEE Trans. Comput.-aided Des. Integ. Circ. Syst.* (2020). DOI: https://doi.org/10.1109/TCAD.2020.3019984

[34] Yangdi Lyu and Prabhat Mishra. 2020. Scalable concolic testing of RTL models. *IEEE Trans. Comput.* (2020). DOI: https://doi.org/10.1109/TC.2020.2997644

[35] Prabhat Mishra, Swarup Bhunia, and Mark Tehranipoor. 2017. *Hardware IP Security and Trust* (1st ed.). Springer Publishing Company, Incorporated.

[36] Melanie Mitchell. 1996. *An Introduction to Genetic Algorithms*. The MIT Press, Cambridge, MA.

[37] S. Narasimhan, D. Du, R. S. Chakraborty, S. Paul, F. G. Wolff, C. A. Papachristou, K. Roy, and S. Bhunia. 2013. Hardware Trojan detection by multiple-parameter side-channel analysis. *IEEE Trans. Comput.* 62, 11 (Nov. 2013), 2183–2195. DOI: https://doi.org/10.1109/TC.2012.200

[38] A. N. Nowroz, K. Hu, F. Koushanfar, and S. Reda. 2014. Novel techniques for high-sensitivity hardware Trojan detection using thermal and power maps. *IEEE Trans. Comput.-aided Des. Integ. Circ. Syst.* 33, 12 (2014), 1792–1805.

[39] Zhixin Pan and Prabhat Mishra. 2021. Automated test generation for hardware Trojan detection using reinforcement learning. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC'21)*.

[40] Zhixin Pan, Jennifer Sheldon, and Prabhat Mishra. 2020. Test generation using reinforcement learning for delay-based side-channel analysis. In *Proceedings of the IEEE/ACM International Conference on Computer-aided Design (ICCAD'20)*.

[41] Jim Plusquellic and Fareena Saqib. 2018. *Detecting Hardware Trojans Using Delay Analysis*. Springer International Publishing, Cham, 219–267. DOI: https://doi.org/10.1007/978-3-319-68511-3_10

[42] I. Pomeranz and S. M. Reddy. 2004. A measure of quality for n-detection test sets. *IEEE Trans. Comput.* 53, 11 (Nov. 2004), 1497–1503.

[43] R. Rad, J. Plusquellic, and M. Tehranipoor. 2010. A sensitivity analysis of power signal methods for detecting hardware Trojans under real process and environmental conditions. *IEEE Trans. Very Large Scale Integ. (VLSI) Syst.* 18, 12 (Dec. 2010), 1735–1744.

[44] S. K. Rao, D. Krishnankutty, R. Robucci, N. Banerjee, and C. Patel. 2015. Post-layout estimation of side-channel power supply signatures. In *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST'15)*. 92–95. DOI: https://doi.org/10.1109/HST.2015.7140244

[45] E. M. Rudnick, J. H. Patel, G. S. Greenstein, and T. M. Niermann. 1997. A genetic algorithm framework for test generation. *IEEE Trans. Comput.-aided Des. Integ. Circ. Syst.* 16, 9 (Sep. 1997), 1034–1044. DOI: https://doi.org/10.1109/43.658571

[46] Sayandeep Saha, Rajat Subhra Chakraborty, Srinivasa Shashank Nuthakki, Anshul, and Debdeep Mukhopadhyay. 2015. *Improved Test Pattern Generation for Hardware Trojan Detection Using Genetic Algorithm and Boolean Satisfiability*. Springer Berlin, 577–596.

[47] H. Salmani, M. Tehranipoor, and J. Plusquellic. 2012. A novel technique for improving hardware Trojan detection and reducing Trojan activation time. *IEEE Trans. Very Large Scale Integ. (VLSI) Syst.* 20, 1 (Jan. 2012), 112–125. DOI: https://doi.org/10.1109/TVLSI.2010.2093547

[48] M. Tehranipoor and F. Koushanfar. 2010. A survey of hardware Trojan taxonomy and detection. *IEEE Des. Test Comput.* 27, 1 (Jan. 2010), 10–25. DOI: https://doi.org/10.1109/MDT.2010.7

[49] Adam Waksman, Matthew Suozzo, and Simha Sethumadhavan. 2013. FANCI: Identification of stealthy malicious logic using Boolean functional analysis. In *Proceedings of the ACM SIGSAC Conference on Computer Communications Security*. ACM, New York, NY, 697–708.

[50] F. Wolff, C. Papachristou, S. Bhunia, and R. S. Chakraborty. 2008. Towards Trojan-free trusted ICs: Problem analysis and detection scheme. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE'08)*. 1362–1365. DOI: https://doi.org/10.1109/DATE.2008.4484928

[51] T. F. Wu, K. Ganesan, Y. A. Hu, H. P. Wong, S. Wong, and S. Mitra. 2016. TPAD: Hardware Trojan prevention and detection for trusted integrated circuits. *IEEE Trans. Comput.-aided Des. Integ. Circ. Syst.* 35, 4 (Apr. 2016), 521–534.
[52] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor. 2016. Hardware Trojans: Lessons learned after one decade of research. *ACM Trans. Des. Autom. Electron. Syst.* 22, 1 (May 2016). DOI: https://doi.org/10.1145/2906147
[53] Z. Zhou, U. Guin, and V. D. Agrawal. 2018. Modeling and test generation for combinational hardware Trojans. In *Proceedings of the IEEE 36th VLSI Test Symposium (VTS'18)*. 1–6.

Received May 2020; revised September 2020; accepted November 2020