Scalable Test Generation for Trojan Detection using Side Channel Analysis

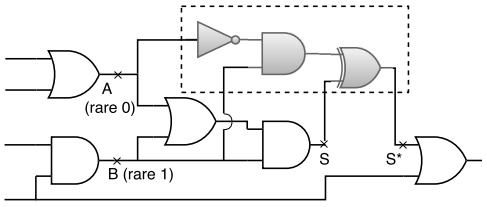
Yuanwen Huang*, Swarup Bhunia†, Prabhat Mishra*
*Department of Computer and Information Science and Engineering, University of Florida, USA
†Department of Electrical and Computer Engineering, University of Florida, USA

***Abstract*—Hardware Trojan detection has emerged as a critical challenge to ensure security and trustworthiness of integrated circuits. A vast majority of research efforts in this area has utilized side-channel analysis for Trojan detection. Functional test generation for logic testing is a promising alternative but it may not be helpful if a Trojan cannot be fully activated or the Trojan effect cannot be propagated to the observable outputs. Side-channel analysis, on the other hand, can achieve significantly higher detection coverage for Trojans of all types/sizes, since it does not require activation/propagation of an unknown Trojan. However, they have often limited effectiveness due to poor detection sensitivity under large process variations and small Trojan footprint in side-channel signature. In this paper, we address this critical problem through a novel side-channel-aware test generation approach, based on a concept of Multiple Excitation of Rare Switching (MERS), that can significantly increase Trojan detection sensitivity. The paper makes several important contributions: i) it presents in detail a scalable statistical test generation method, which can generate high-quality testset for creating high relative activity in arbitrary Trojan instances; ii) it analyzes the effectiveness of generated testset in terms of Trojan coverage; and iii) it describes two judicious reordering methods that can further tune the testset and greatly improve the side channel sensitivity. Simulation results demonstrate that the tests generated by MERS can significantly increase the Trojans sensitivity, thereby making Trojan detection effective using side-channel analysis.**

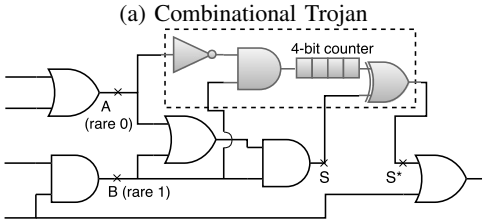
I. Introduction

Hardware Trojan attacks relate to malicious modifications in the design of Integrated Circuits (ICs) at different stages of the design or fabrication process [2]. An adversary can introduce these modifications in a design in order to cause disruption in normal functional behavior and/or to leak secret information from a chip during operation in field. Since the threat of hardware Trojan in the form of a malicious implant in a design came into light about a decade ago through an US Department of Defense announcement [3], it has triggered vast body of research activities in threat analysis as well as design/validation solutions to evaluate this threat and protect against it. Hardware Trojan attacks are also being increasingly recognized in the semiconductor industry as a serious concern in terms of security and trustworthiness of ICs.

A Trojan is expected to be covert and difficult to detect, i.e. an intelligent adversary will likely insert a Trojan circuit in a way that evades detection during post-manufacturing functional/parametric testing, but manifests itself during long hour of in-field operation. This can be achieved by externally triggering its operation or by making it dependent on rare circuit conditions inside an IC. The condition of Trojan activation as commonly referred to as *trigger condition*, which can be purely combinational or sequential. The latter is related to the clock or a sequence of rare events in the state elements (e.g. flip-flops of registers). The internal circuit nodes affected by a Trojan activation are referred to as *payload* of a Trojan. Fig. 1 shows some example Trojan circuits including a combinational and a sequential Trojan. For example, a Trojan circuit could be triggered only when a data bus attains a unique rare value or when the number of times it attains the rare value equals to a particular count. The malicious effects of Trojan payloads can range from passive, such as leakage of secret information, to altering the original functionality of the chip in a critical or destructive fashion.



(a) Combinational Trojan



(b) Sequential Trojan

Fig. 1: Example of a combinational and a sequential Trojan with triggers (A, B) from rare internal nodes, and payload S.

Protection against hardware Trojan attacks can be accomplished in two broad ways: (1) design-for-security (DfS) techniques that make Trojan insertion difficult or make a Trojan easily detectable through post-silicon testing; and (2) manufacturing test approaches that aim at detecting an arbitrary Trojan by observing its effect into a circuit's operational behavior. The first class of techniques, primarily relies on different types of hardening approaches - e.g. insertion of dummy cells into empty spaces in a circuit layout; or key-based obfuscation of a design that makes malicious alteration by an adversary provably hard. DfS techniques, however, come at the cost of additional design, verification, and test time, as well as hardware overhead. For example, key-based obfuscation, even though is capable of providing high level of robustness against

Trojan attacks, come at a cost of 10% or more area overhead [4]. More importantly, design solutions, however, only work for new designs and not legacy designs, and hence has limited applicability. Hence, efficient test/validation approaches that can provide high level of confidence regarding IC trustworthiness in presence of Trojan threat provides an attractive solution to the IC manufacturers.

Existing test generation solutions for hardware Trojan detection can be broadly classified into two categories: 1) logic testing and 2) side-channel analysis. In logic testing approach, directed tests are generated to activate rare events in a circuit and propagate the malicious effect of a Trojan in logic values to observable outputs. Such approaches are known to be more effective in detecting ultra-small Trojans (typically a few gates in size). The main challenge with logic testing approaches, however, is the difficulty to trigger a Trojan and observe its effect, particularly in the presence of complex sequential Trojans, and the inordinately large number of possible Trojan instances that an adversary can exploit. On the other hand, side-channel analysis approaches depend on the measurement and analysis of physical side-channel parameters like power signature or path delay of an IC in order to identify a structural change in the design. Unlike logic testing, these approaches do not require Trojan activation in order to detect them. Side-channel analysis (SCA), primarily based on supply current, has been extensively investigated by large number of research groups and various solutions to increase the signal-to-noise (SNR) have been proposed. A disadvantage of SCA arises from the large process variations (e.g. 20X leakage power and 30% delay variations in 180nm technology [5]) which can potentially mask the minute effect of a Trojan in the measured side-channel parameter. A solution to the sensitivity problem can be achieved by judicious test generation approach that aims at maximizing the sensitivity for an arbitrary Trojan in unknown circuit location. In the remainder of the paper, we focus on transient current or power as our side-channel parameter of interest. Some of the concepts however can be applied to other side-channel parameters. To maximize sensitivity of a given Trojan, one needs to amplify activity inside the Trojan circuit and simultaneously minimize the background activity (i.e. activity in the original circuit). We present a novel statistical test generation framework that can maximize the detection sensitivity for an arbitrary Trojan.

The goal of our work is to generate efficient test vectors for Trojan detection using side-channel analysis. We use the relative switching of the Trojan with respect to the whole circuit to indicate the sensitivity of the side channel signals. The statistical test patterns can maximize relative Trojan detection sensitivity under any process noise. Process variation is not expected to affect our side channel sensitivity computation since we consider switching activity instead of actual current or power values. The proposed method can be combined with any existing process calibration approaches (such as one in [24] or [25]) to minimize the false positives/negatives and maximize Trojan coverage. The following are the major contributions of this paper:

1) It presents, for the first time in our knowledge, a statistical test generation approach for increasing side-channel analysis based Trojan detection sensitivity. The proposed approach can be applicable to any transient current based Trojan detection approach.
2) The methodology, referred to as MERS (Multiple Excitation of Rare Switching) for statistical test generation, is shown to derive a compact testset that can trigger each of the rare nodes to satisfy *rare switching* for multiple times.
3) Reordering methods are proposed to reduce the total switching of the circuit and thus further increase the sensitivity of side channel analysis. A simple and low-cost method based on Hamming distance of input vector pairs is introduced to reorder the tests. We also develop another simulation based method to more effectively balance switching in rare nodes and the total switching.
4) Design partition methods are proposed for scalable test generation for large designs. By functionally and/or structurally partitioning the design into regions, we apply targeted test generation on each region and significantly improve the side channel sensitivity for large designs.

Our side-channel based approach is targeted towards detecting unknown Trojans, which means it will remain equally effective even if the adversary is aware of the proposed method. This is due to the following two reasons: (1) the proposed test generation method is statistical in nature - so, unlike conventional deterministic test approaches, it maximizes the activation probability for arbitrary Trojans designed with any trigger condition; and (2) it maximizes the detection sensitivity of unknown Trojans, however "stealthy", by amplifying its effect in side-channel signature. Our simulation platform inserts large number of arbitrary Trojans in a design and shows that the proposed approach is highly effective in detecting them.

The rest of the paper is organized as follows. Section II presents related work in side channel analysis and functional test generation for Trojan detection. Section III describes our proposed MERS test generation algorithm and the test reordering algorithms to improve sensitivity of side channel analysis. Section IV describes the experimental setup and presents results on a set of ISCAS benchmarks with detailed analysis. Section V presents results for two large designs (AES cipher and DLX processor). Section VI concludes the paper.

II. Related Work

The underlying assumption for Trojan insertion is that an adversary is fully aware of the design functionality and therefore can hide the Trojan in a hard-to-find place. One way to address this issue is to obfuscate [4] or encrypt [15] the design such that the adversary cannot figure out the actual functionality and therefore cannot insert the Trojan in a covert manner. Unfortunately, smart attacker can effectively bypass both obfuscation [16] and encryption [33] methods. A promising direction is to develop efficient techniques for hardware Trojan detection. Prior research on Trojan detection can be classified into two broad categories: side-channel analysis and functional test generation.

Side-channel analysis approaches are based on analysis of side-channel signatures such as circuit transient current

[9][10][14], power consumption [12][13], path delay [11], or intermediate values from debug infrastructure [19]. The basic idea is to compare the side-channel signature with the pre-characterized golden value for a Trojan-free IC (or a model of the IC). If the observed value of the measured parameter differs by more than a threshold from the golden value, the presence of a Trojan is suspected. Unfortunately, side-channel analysis has a common issue, i.e., the sensitivity of side-channel signatures is susceptible to thermal and process variations. Therefore, it would be difficult to detect small combinational Trojans. In this paper, we also rely on transient current (switching activity) to identify Trojan.

Compared with [9][10][14], our approach can greatly increase the side-channel sensitivity of Trojan of any type or size, because we take advantage of functional testing. In other words, our test vectors are generated in a statistical way, and they are more effective in creating switching in Trojan, as well as reducing background switching. The approach proposed by Banga and Hsiao [9] partitions a design into circular regions (with a center and radius) for side channel analysis. A region is a group of flip-flops along with combinational gates connecting them. However, there are two major drawbacks with their partitioning approach. First, there are thousands of regions identified even for a small ISCAS89 benchmark s3271. It may be infeasible to generate targeted tests for each of the regions in large designs. Next, regions identified by their approach may overlap with each other, while our approach can ensure the regions are disjoint. Banga et al. proposed in [10] to partition a circuit into flip-flop groups based on structural connectivity. However, the scalability of the approach to large designs with datapaths and control structure is limited. Moreover, it is difficult to judge their effectiveness since they only tested on very small circuits. Salmani et al. [14] proposed a layout-aware approach for improving localized switching to detect Trojan. Their approach is based on reordering the scan cells (flip-flops) in the chip, which is orthogonal to our approach of test generation for improving switching.

Another category of Trojan detection approaches is to generate functional test patterns that are likely to fully activate the Trojans. These approaches can overcome the effect of thermal and process variations on side-channel signals. They rely on the fact that an adversary will choose a trigger condition for the Trojan using a set of rare nodes. Various approaches tried to maximize the rare node activation to increase the likelihood of activating Trojans. ATPG for Trojan detection is investigated in [6][32]. A major problem with ATPG based Trojan detection methods is the scalability issue. ATPG can be used to activate a Trojan if all the triggers are known. However, this is not feasible for Trojan detection since Trojans are likely to have unknown number of triggers hidden at stealthy locations. It would be practically infeasible to use ATPG to test all possible trigger conditions. MERO [7] takes the advantage of N-detect test [23] to maximize the trigger coverage by activating the rare nodes. The test generation ensures that each of the nodes gets activated to their rare values for at least N times. It is shown that if N is sufficiently large, a Trojan with trigger condition based on these rare nodes is likely to be activated by the generated test set. Saha et al. [8] improve the test pattern generation of MERO [7] by using genetic algorithm and Boolean satisfiability for ATPG. Their approach could more effectively propagate the payload of possible Trojan candidates. A design-for-test (DFT) infrastructure technique by Salmani et al. [21] inserts dummy flip-flops to increase the transition probability of low-transition nets, and therefore increases the side-channel sensitivity for Trojan detection. Zhou et al. [22] further improved their approach by selecting the most beneficial nets to insert dummy flip-flops based on fanout analysis. Farahmandi et al. [20] attempted to localize Trojan using symbolic algebra from a formal verification approach, while it is not scalable to large circuits.

Direct application of test generation approaches is not suitable for improving side-channel sensitivity for Trojan detection. The objective of increasing side-channel sensitivity is very different from the ones in both MERO [7] as well as its enhanced version by Saha et al. [8]. Unlike these existing techniques, our proposed approach requires the creation of a pair of test vectors to maximize switching in rare nodes. Our algorithm creates multiple excitation of rare switching which is important in making side-channel based Trojan detection effective. The basic concept is presented in our conference paper [1], but it does not provide a scalable test generation framework for different DFT structures. Moreover, we also try to simultaneously minimize the background switching to maximize the relative switching.

Our test generation method also originates from N-detect test. Compared with MERO [7], which focuses on logic testing with N-detect test, we target generating vectors for side-channel analysis. The primary difference is that MERO tries to assign rare values (0 or 1), whereas our approach tries to assign rare transitions ($0 \rightarrow 1$, or $1 \rightarrow 0$). Specifically, they have three important differences. First, MERO's goal is to generate tests which can fully trigger the Trojan and observe the propagated Trojan effect. Our algorithm aims at creating more switching in possible Trojan triggers to greatly improve the side-channel sensitivity and expose hideous Trojans. Next, MERO's approach is mostly limited to combinational Trojans with smaller number of triggers. MERO is not effective for sequential Trojans or larger Trojans. Our approach focuses on switching of rare nodes, which makes it effective to any type/size of Trojans hidden at any location. Finally, by utilizing functional and structural partitioning, our approach is scalable to large designs with a large number of rare nodes or possible triggering conditions.

III. Scalable Test Generation for Side-Channel Aware Trojan Detection

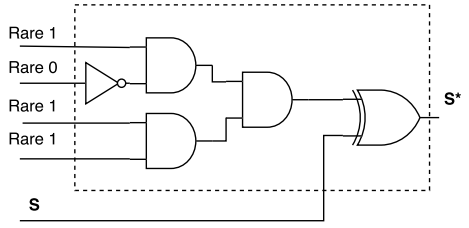
In this section, we present the proposed methodology for side-channel aware test generation in detail. The methodology is based on the concept of statistically maximizing the switching activity in all the rarely triggered circuit nodes.

The effectiveness of a test pattern for side channel analysis is measured in two ways: (1) the ability to create most switching inside a Trojan or to activate a Trojan; (2) the ability to create high Trojan-to-circuit switching. We measure *DeltaSwitch* as the switching introduced by the Trojan,

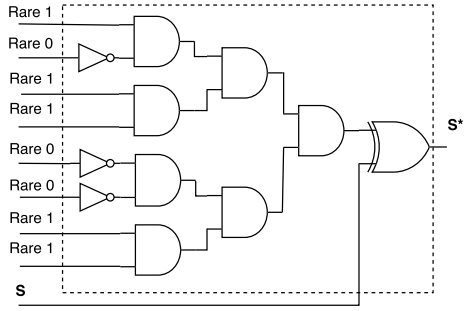
which is the difference of number of switches between the golden circuit and the Trojan-infected circuit. We measure *RelativeSwitch* as the ratio of *DeltaSwitch* to the total number of switches (*TotalSwitch*) in the golden circuit. An effective test vector should be capable of creating large *DeltaSwitch*, and more importantly it should create large *RelativeSwitch*, as it is directly related to the sensitivity for side channel analysis.

$$RelativeSwitch = DeltaSwitch/TotalSwitch \quad (1)$$

The major challenges for generating high-quality test vectors are as follows: (1) we are not sure of the location where the Trojan is inserted in the circuit; (2) the Trojan is stealthy and has very low activity when it is not triggered. These characteristics have made random tests not effective in magnifying the side channel signal for Trojan detection. Fig. 2 shows two example Trojan instances. The 4-trigger Trojan will only be activated by the rare combination 1011 and the 8-trigger Trojan will only be activated by the rare combination 10110011. If the possibility of each rare node to take its rare value is 0.1, the probability to have these two Trojans fully triggered is 10^{-4} and 10^{-8}, respectively.



(a) A 4-trigger Trojan



(b) An 8-trigger Trojan

Fig. 2: Trojans with rare nodes as trigger conditions.

A. Overview

In this subsection, we provide the overview of the workflow for scalable test generation for side-channel aware Trojan detection. As shown in Fig. 3, we first simulate the circuit to get the rare nodes, which have low probability to be 0/1. We partition the design into regions, apply our MERS approach to generate tests and also reorder the tests for each region. The vectors from all regions are combined into a test suite that can create high relative switching for arbitrary Trojans in the design.

Rare Nodes Identification: In our experiments, we simulate the circuit with 100,000 random vectors and note down the probability of values for internal nodes. Nodes with probability less than the rare threshold are identified as rare nodes. These rare nodes are the candidates for Trojan triggers. We sample stealthy Trojans with triggers from the rare nodes for evaluation of our test generation approach.

Design Partitioning: A major challenge of large designs is that the supply current of a golden chip for a high-activity vector can be very large compared to the additional current consumed by a small Trojan. If we can carefully partition a circuit into nearly-isolated regions (i.e. with low connectivity between them), we can more effectively generate tests for each region. After partitioning the design, test generation can target on the rare nodes inside each region, but also try to avoid creating too many background switching.

Test Generation: Our test generation approach (MERS) is based on creating a set of test vectors for each candidate rare node individually to have *rare switching* multiple (at least N) times. Our approach utilizes the principle of N-detect [23] tests to increase the likelihood of partially or fully activating a Trojan. MERS can generate a high-quality testset for these rare nodes individually to have rare switching for N times. If N is sufficiently large, a Trojan with triggering conditions from these rare nodes is likely to have high switching activity even though it might not be fully activated.

Test Reordering: The order of test vectors matters as we are counting the switching between two vectors. Our goal is to further improve the side channel sensitivity. The challenge is to keep the high-quality in creating switching on rare nodes, and at the same time to reduce the background switching. We introduce Hamming-distance based reordering and simulation based reordering to resolve this challenge.

Testset Evaluation: In our experiments, we insert a Trojan into the design, then apply all test vectors in the combined testset. The side-channel sensitivity is reported as the maximum relative switching of the testset. To show that our approach has good coverage on Trojans hidden at different locations, we experimented on 1000 Trojan samples to evaluate the effectiveness of testsets. From a pool of potential rare Trigger nodes, a Trojan of given size is created by randomly choosing the trigger nodes and the payload. After that we verify if this trigger condition and payload make a functionally valid Trojan, i.e. it can be activated using a valid input condition and its malicious effect propagates to any observable output. Thus, we consider only valid random Trojans in our evaluation. The statistical nature of MERS ensures that even if an adversary chooses different locations or trigger conditions for inserting Trojans, the test set can maximize the detection sensitivity for them.

Algorithm 1 shows the steps for scalable test generation on large designs. We first simulate the circuit to identify rare nodes and generate stealthy Trojan samples. If design partitioning is enabled, the design is partitioned according to natural boundaries based on functionality. If the design has no such natural boundaries, or the partitioned region is till too large, we can perform structural partitioning based on circuit connectivity. For each region, we apply our high-quality test generation approach MERS (Algorithm 2), followed by test reordering (Algorithm 3 or 4) for further improvement in side channel sensitivity. Finally, the test patterns from all region will be combined together to evaluate the effectiveness of our

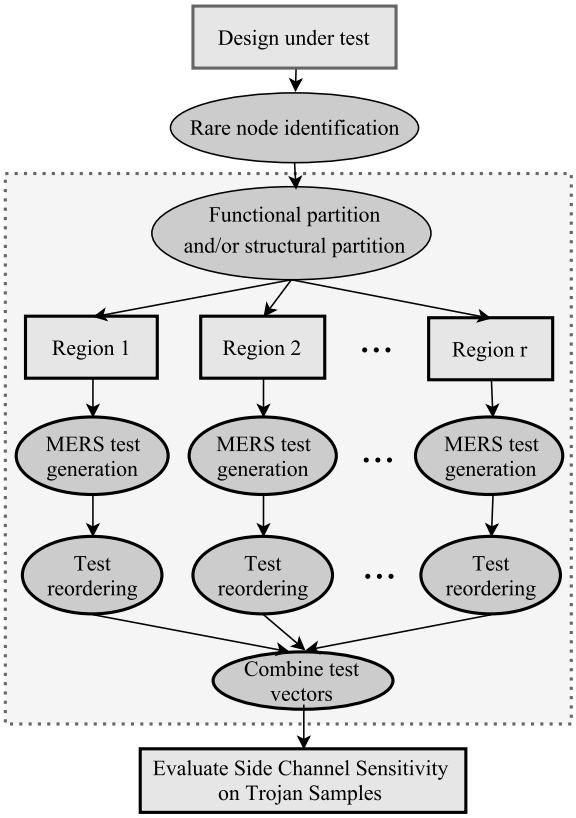


Fig. 3: Scalable test generation for side-channel analysis based Trojan detection.

test generation approach on the Trojan samples.

Algorithm 1: SCALABLE TEST GENERATION

Input: Circuit under test
Output: Test patterns for Trojan Detection

```
// Rare nodes identification
Simulation to identify nodes with low probability
Generate Trojan samples with triggers from rare nodes
// Design Partitioning
if Design partition is enabled then
    if Design has natural submodules then
        Functional partition into regions.
    end
    if Design (or any region) is large then
        Structural partition based on connectivity.
    end
end
// Test Generation
for each region do
    Test generation with MERS (Algorithm 2)
    Test reordering with Algorithm 3 or 4
end
// Evaluation of Test Patterns
Combine the testsets from all regions
Evaluate side-channel sensitivity on Trojan samples
```

B. Design Partitioning

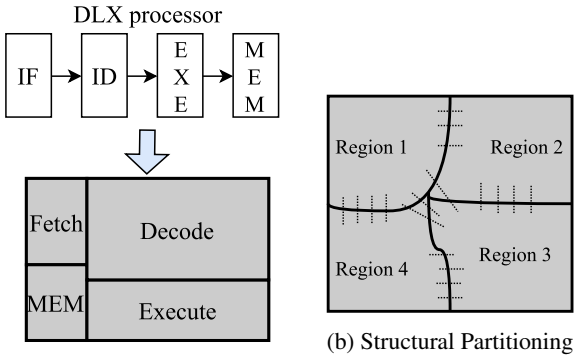
There are at least three advantages of dividing a large design into smaller regions. (1) For a designated region, region-based MERS (Algorithm 2) will only target the rare nodes in that region to have rare switching for N times. The quality of tests is likely to improve and many rare nodes can achieve rare switching a lot more than N times. (2) The rare nodes outside of the designated region will be ignored. Since the test generation process doesn't try to switch those rare nodes, it is likely to create fewer switching in the outside regions and reduce the background switching. (3) Assuming that the sequential circuits is equipped with scan-chains, we can shift 0's into the flip-flops (the pseudo primary inputs) that are outside of the designated region. This can further reduce the background switching of other regions.

The partitioning approach should divide the design into regions, which have minimum inter-connections between them. In other words, we want each region to be functionally independent or have as few connections as possible to other regions, so that the test generation process can increase the activity of one region (or few regions) while minimizing the activity of all others. A complex circuit under test usually comprises of several functional modules (or regions), which are interconnected according to their input/output dependencies. For the example in Fig. 4a, the DLX processor has four pipeline stages (IF, ID, EXE, and MEM). It can be naturally partitioned into four regions according to the functional modules: Fetch, Decode, Execute, and Memory. We can fill the pipeline such that the different pipeline stages are activated one at a time during test generation. An alternative to functional partitioning is structural partitioning as shown in Fig. 4b. Structural partitioning is the only choice when functional partitioning is not possible (e.g., flattened netlist). Structural partitioning can use hypergraph partitioning approach [27] or any other region-based partitioning approaches [9]. It is important to note that our approach can effectively combine both partitioning techniques. For example, after functional partitioning has been performed on DLX processor (Fig. 4a), structural partitioning (Fig. 4b) can be applied on the Decode module (accounting for 71% of the whole design area) to further partition it into smaller regions.

We use structural partitioning to improve the side channel sensitivity for the three ISCAS sequential benchmarks: s13207, s15850, and s35932. Since these benchmarks are flat netlists (i.e. we cannot easily identify any functional regions), we use hypergraph partitioning [27] to find relatively isolated regions. The circuit is transformed into a hyper-graph, where each gate is a vertex and the set of gates sharing an edge is a hyper-edge. The goal of partitioning is to divide the graph into two partitions of roughly same size[1] (each contains 45% to 55% of the total number of vertices). The constraint is that the minimal number of hyper-edges will be *cut* by the partition process. This constraint is to ensure that the vertices inside each region have high connectivity, while the connectivity

[1]We have performed structural partitioning [27] with partition factor $p = 50\%$, which generally achieves better SCS than a skewed partition. If the designer can afford the cost, different partition factor p can be explored.

between regions is minimal. This is a well-studied problem in hypergraph, and we used the tool from [27] to satisfy this purpose. The whole design is first partitioned into two regions of almost equal size with minimal hyper-edge cuts. Each of the two regions can be partitioned into two smaller sub-regions, and so on. In other words, we can partition the design in a recursive manner to have 2 regions, 4 regions and 8 regions.



(a) Functional Partitioning

(b) Structural Partitioning

Fig. 4: (a) A design with functional modules can be naturally partitioned. (b) A flat design can be structurally partitioned to find regions with minimum inter-connections being cut.

C. Multiple Excitation of Rare Switching (MERS)

The basic idea of MERS is that if we can make a rare node switch N times where N is sufficiently large, it significantly improves the chance of switching in a Trojan associated with that rare node. The **rare switching** in our algorithm specially refers to a rare node switching from its non-rare value to its rare value. The reason to choose this criteria is two-fold: (1) it is more difficult to switch from non-rare to rare value than from rare to non-rare value; (2) it defines the switching between the previous vector and the current vector, and it usually helps to create an extra switching between the current vector and the next vector. This will increase the probability of switching of a Trojan which has rare nodes as its trigger conditions. Our approach is also applicable to sequential Trojans, which requires the rare condition to occur a certain number of times to be fully triggered.

Algorithm 2 shows the steps of MERS to generate high quality tests for creating switching in rare nodes, so as to assist side channel analysis for hardware Trojan detection. The algorithm is fed with the golden circuit netlist, the list of random test patterns (V) and a list of rare nodes (R) (which is obtained by random vector based circuit simulation beforehand). First, we simulate each random pattern and count the number of rare nodes (R_V) that take their rare values. We sort the random patterns in descending order of R_V, which means that the vector with ability to activate the most number of rare nodes goes first. Next, we initialize the rare switching counter S_i for each rare node to 0. In the next step, we mutate vectors from the random pattern set to generate high quality tests. We mutate the current vector one bit at a time and we accept the mutated bit only if the mutated vector can increase the number of nodes to have rare switching. In this step, only those rare nodes with $R_S < N$ are considered. The mutation process repeats until each rare node has achieved at least N rare switches. The output of the test generation process is a compact set that improves the switching capability in rare nodes, compared to random patterns. The complexity of the algorithm is $O(n * m)$, where n is the total number of test vectors mutated during the process, and m is the number of bits in primary inputs. The runtime to generate MERS tests can be found in Table I.

Algorithm 2: MULTIPLE EXCITATION OF RARE SWITCHING (MERS)

Input: Circuit netlist (targeted region), rare switching requirement (N), the list of rare nodes ($R = \{r_1, r_2, ..., r_m\}$), the list of random patterns ($V = \{v_1, v_2, ..., v_n\}$)
Output: MERS test patterns (T)

```
for each random vector v in V do
    Simulate the circuit with the input vector v
    Count the number of nodes (R_V) in R with their rare values satisfied
end
Sort vectors in V in descending order of R_V
for each node r_i in R do
    Set its rare switching counter (S_i) to 0
end
Initialize previous vector t_p as a vector of all 0's
for each vector v_j in V do
    Simulate the circuit with vector pair (t_p, v_j)
    Count the number of rare switches (R_S)
    Set v'_j = v_j
    for each bit in v'_j do
        Mutate the bit and re-simulate the circuit with vector pair (t_p, v'_j)
        Count the number of rare switches (R'_S)
        if R'_S > R_S then
            Accept the mutation to v'_j
        end
    end
    Update S_i for all nodes in R due to vector v'_j
    if v'_j increases S_i for at least one rare node then
        Add the mutated vector v'_j to T
        Set t_p = v'_j
    end
    if S_i ≥ N for all nodes in R then
        Break
    end
end
return MERS test patterns T
```

The testset generated by MERS is expected to be very effective in increasing the likelihood of rare nodes to switch and thus increasing the activities in Trojans. In other words, MERS testset is capable of maximizing the DeltaSwitch (the numerator in Equation 1). MERS testset is already a very high quality testset in terms of criteria for DeltaSwitch. However, MERS testset also creates more switching in other parts of

the circuit, when it is making efforts to switch rare nodes. This characteristic of increased TotalSwitch would be further illustrated in the Section IV. In order to maximize relative switching, we need to have TotalSwitch in control as well. In the following subsections, we propose two methods to tune the MERS testset, so that it can: (1) still be effective for DeltaSwitch, (2) reduce TotalSwitch and improve the effectiveness for RelativeSwitch. The first method is a heuristic approach based on hamming distance of test vectors, which can reduce the total switching. The second one is simulation based, in which we try to balance the rare switching and the total switching while we explore all the candidate vectors.

Algorithm 3: TESTS REORDERING BY HAMMING DISTANCE (MERS-h)

Input: List of Test Patterns ($T_{orig} = \{t_1, t_2, ..., t_n\}$) produced by Algorithm 1
Output: Improved Test Patterns (T_{hamm})

Initialize $T_{hamm} = \{\}$
Initialize previous test t_p as a vector of all 0's
while T_{orig} *is not empty* **do**
    $min_{dist} = int\_max$
    $best_{idx} = -1$
    for all *remaining tests* t_j *in* T_{orig} **do**
        if $min_{dist} > hamming\_dist(t_p, t_j)$ **then**
            $min_{dist} = hamming\_dist(t_p, t_j)$
            $best_{idx} = j$
        end
    end
    Add $t_{best_{idx}}$ to the end of T_{hamm}
    Remove $t_{best_{idx}}$ from T_{orig}
    Update $t_p = t_{best_{idx}}$
end
return T_{hamm}

D. Test Reordering

1) Hamming Distance based Reordering: If two consecutive input vectors have the same values in most bits, it is very possible that the internal nodes will also have a lot of values in common. A simple heuristic to reduce total switching in circuit is to have similar input vectors. We use the Hamming distance between two vectors to represent the similarity. Algorithm 3 shows our approach to reorder the testset by Hamming distance. The algorithm is a greedy approach to explore all candidate vectors and take the best one in terms of Hamming distance. We first check the Hamming distances between the previous vector and all the remaining vectors, then we select the vector which has the minimum Hamming distance as the next vector. The time complexity of Algorithm 3 is $O(n^2)$, where n is the testset size. Fortunately, it is of low cost to calculate the Hamming distance between two input vectors, so the actual run-time is very short.

2) Simulation based Reordering: The reordering problem to improve the relative switching is actually a multi-objective optimization problem: maximize the $DeltaSwitch$ and minimize the $TotalSwitch$ as in Equation 1. We do not know the

Algorithm 4: TESTS REORDERING BY SIMULATION (MERS-s)

Input: List of Test Patterns ($T_{orig} = \{t_1, t_2, ..., t_n\}$) produced by Algorithm 1
Output: Improved Test Patterns (T_{sim})

Initialize $T_{sim} = \{\}$
Initialize previous test t_p as a vector of all 0's
while T_{orig} *is not empty* **do**
    $max_p = int\_min$
    $best_{idx} = -1$
    for all *remaining tests* t_j *in* T_{orig} **do**
        Simulate the circuit with vector pair (t_p, t_j)
        Count the number of RareSwitch and TotalSwitch
        $profit = C * RareSwitch - TotalSwitch$
        if $max_p < profit$ **then**
            $max_p = profit$
            $best_{idx} = j$
        end
    end
    Add $t_{best_{idx}}$ to the end of T_{sim}
    Remove $t_{best_{idx}}$ from T_{orig}
    Update $t_p = t_{best_{idx}}$
end
return T_{sim}

$DeltaSwitch$, because the location and type of the Trojan is unknown. However, rare switching between two vectors is a good indicator for $DeltaSwitch$, which means a large number of rare switching would imply a large $DeltaSwitch$ in Trojan. We redefine the optimization goal as to maximize the rare switching and minimize the total switching at the same time between vector pairs. We formalize the problem as shown in Equation 2. We need to explore the best weights to balance between the two objectives:

$$maximize \quad (w_1 * RareSwitch - w_2 * TotalSwitch) \quad (2)$$

We propose an approach as shown in Algorithm 4 based on real simulation of the test vectors to maximize the combined objective. We introduce a concept of $profit$ to indicate the fitness of a test vector to follow the previous test vector. $profit$ is defined as $(C * RareSwitch - TotalSwitch)$, where C is the ratio of two weights w_1 and w_2. It is meant to maximize the rare switching (activity in Trojan circuits) and minimize the total switching of the whole circuit. In the experiment section, we will explore different weight ratios and check the influence of weight ratios on side channel sensitivity.

Algorithm 4 shows our approach to tune the testset by simulation with $profit$ as a reordering criterion. By exhaustively checking the $profit$ between the previous vector and all the remaining vectors, we select the vector which has the maximum $profit$ as the next following vector. The time complexity of Algorithm 4 is $O(n^2)$, where n is the test length. However, it is much slower than Algorithm 3, because it is time-consuming to simulate input vector pairs and calculate $profit$.

In Algorithm 1, we leave the choice of reordering algorithm to the designer. If simulation time is a concern, Algorithm 3 should be chosen. If the designer wants higher side channel sensitivity and can afford longer simulation time, Algorithm 4 is more suitable. It is possible to combine the fast reordering of Algorithm 3 and the exhaustive exploration of Algorithm 4. The complexity of Algorithm 4 is $O(n^2)$, where n is the number of candidate vectors to be reordered. If we can reduce n, we can improve the runtime of Algorithm 4. For example, we can use Algorithm 3 in the first pass. In the second pass, we can divide the test suites in multiple parts and apply Algorithm 4 on each of them. Increasing the number of divisions will reduce the runtime but at the cost of result quality.

IV. Evaluation Results

A. Experimental setup

The test generation framework, including the MERS core algorithms and the evaluation framework, is implemented using C. As shown in Fig. 3, the test generation framework can identify rare nodes, generate MERS testset, further tune the testset, and evaluate the effectiveness of testsets on random Trojans. We evaluated our approach on a subset of ISCAS-85 and ISCAS-89 benchmark circuits, as well as two large designs AES cipher and DLX processor [31]. The sequential circuits are converted into full scan mode. We also implemented the MERO [7] approach with parameter N of 1000 for comparison. We did our experiments on a server with AMD Opteron Processor 6378 (2.4GHz). The runtime for different benchmarks and different methods is shown in Table I. The table also shows the number of rare nodes in each benchmark. We used 0.1 as the rare threshold to select rare nodes. We can see that if we use Algorithm 3 to reorder by Hamming distance, our runtime is about half of MERO on average. If we use Algorithm 4 to reorder by simulation, our runtime is about 7% longer on average. So it is reasonable to say that our test set is more effective than MERO given the similar test generation time.

TABLE I: Runtime comparison for MERO [7], MERS-h and MERS-s, with N=1000, rare threshold = 0.1

Benchmark	**Nodes (rare / total)**	**Runtime (s)**		
		MERO [7]	MERS-h	MERS-s
c2670	63 / 1010	30051.53	13378.1	18296.09
c3540	331 / 1184	9403.11	6106.94	24264.45
c5315	255 / 2485	80241.52	45607.01	84669.78
c6288	45 / 2448	15716.42	4154.93	6957.47
c7552	306 / 3720	160783.37	81431.09	144908.08
s13207	592 / 2504	23432.04	12876.97	41576.67
s15850	679 / 3004	39689.63	20631.58	58084.93
s35932	896 / 6500	29810.49	7335.27	38496.78
Average	**396 / 2857**	**48641**	**23940**	**52157**

B. Evaluation Criteria

When applying a testset to a circuit with Trojan, there are four criteria to evaluate the effectiveness of the testset:

- **AvgDeltaSwitch**: the average delta switch when applying the testset on this Trojan-infected circuit.
- **MaxDeltaSwitch**: the maximum delta switch when applying the testset.
- **AvgRelativeSwitch**: the average relative switch when applying the testset.
- **MaxRelativeSwitch**: the maximum relative switch when applying the testset. We choose this criterion as the **Side Channel Sensitivity** because this directly determines whether a Trojan can be detected through side-channel analysis.

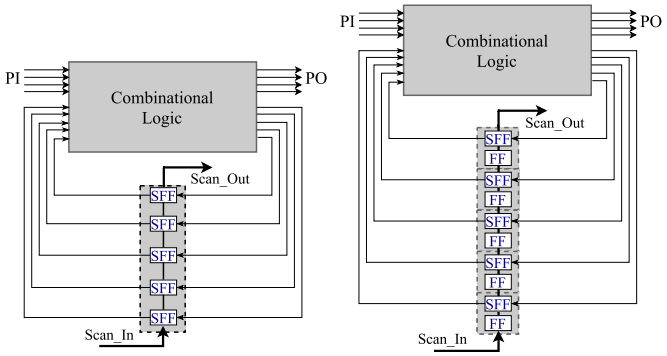
AvgDeltaSwitch and *MaxDeltaSwitch* reflect the activity in Trojan, and *AvgRelativeSwitch* as *MaxRelativeSwitch* reflect the sensitivity of the side channel signal in detecting the Trojan.

As for evaluation of testsets, we would expect a high-quality testset to have a good coverage over all possible Trojans. In our experiments, we apply the testset to 1000 randomly inserted Trojan samples and compute these four values for each Trojan instance. We would then take the average of these four metrics, which would reflect the capability of the testset to enable detection of different Trojans through side-channel analysis. The average $MaxRelativeSwitch$ would be most suitable for Side Channel Sensitivity evaluation, which is to maximize the sensitivity for an arbitrary Trojan in unknown circuit location.

C. Different Scan Modes

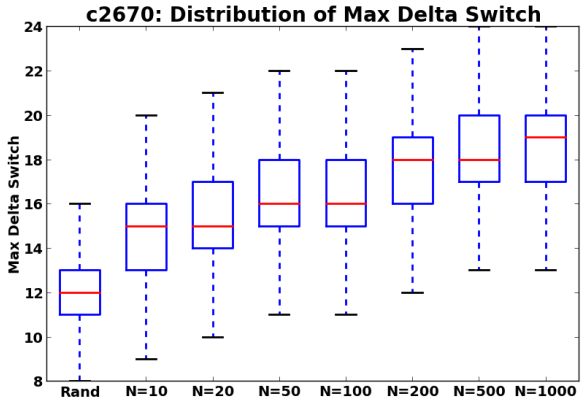
For sequential benchmarks used in our paper, we assume that the sequential gates (i.e. the flip-flops) have full-scan capability during test. The scan flip-flips (SFF) form a scan chain as shown in Fig. 5. The initial states of the circuit can be set by the scan chain. Test vectors feed values to the primary inputs (PI) and the scan flip-flops (also called pseudo-PI). The controllability of the circuit states largely depends on the working mode of the scan chain. The transition test involves applying a vector pair (V_1, V_2) to the circuit. The first vector is to launch the circuit into a desired state. The transitions will be captured after V_2 is applied. V_1 will set the PI values as well as the initial states of circuit through SFFs. V_2 will feed the circuit with a different set of PI values. V_2 may or may not feed the SFFs with new values depending on the scan mode. We measure the number of switching in the circuit for side-channel analysis after V2 is applied.

A conventional scan chain (Fig. 5a) can *Launch-on-Shift* (LoS) and *Launch-on-Capture* (LoC) modes [28]. In both of these two modes, V_2 only feeds the circuit with new values to PIs. The flip-flops will have values either directly from V_1 (shift by 1) or after propagating for one clock cycle. For LoS mode, the second vector V_2 is immediately applied after V_1 is shifted into SFFs. For LoC mode, the second vector V_2 waits for one clock cycle after V_1 is applied to the circuit. An enhanced scan chain (Fig. 5b) can work in *Enhanced* mode [29], [30]. Compared to the LoS and LoC, the enhanced scan chain has one extra redundant flip-flop attached to each of the SFF. After the shifting process, the SFFs hold states for V_1 and the redundant FFs hold states for V_2. This feature enables both V_1 and V_2 to feed arbitrary values to the sequential gates. It comes at the cost of doubling the number of flip-flops. However, it provides high controllability and testability into the sequential circuits. In our experiments, unless explicitly specified, we assume the enhanced scan mode is used for the sequential benchmarks.

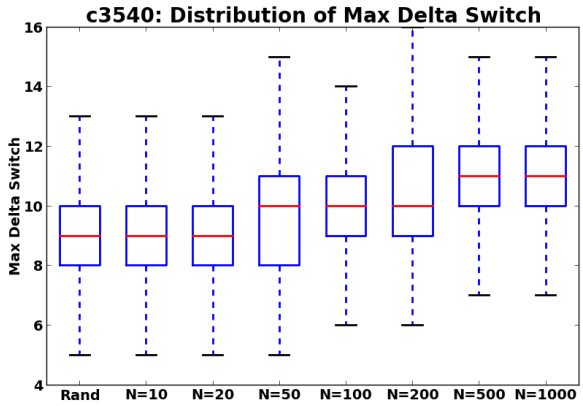


(a) Scan chain (b) Enhanced scan chain

Fig. 5: Scan chains are used to set initial states for flip-flops. A conventional scan chain can work in LoC and LoS modes. An enhanced scan chain can work in Enhanced mode.



(a)

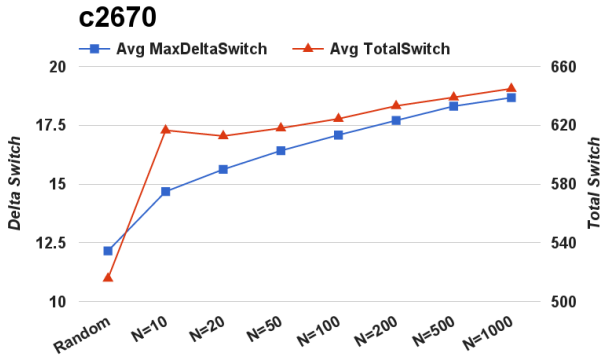


(b)

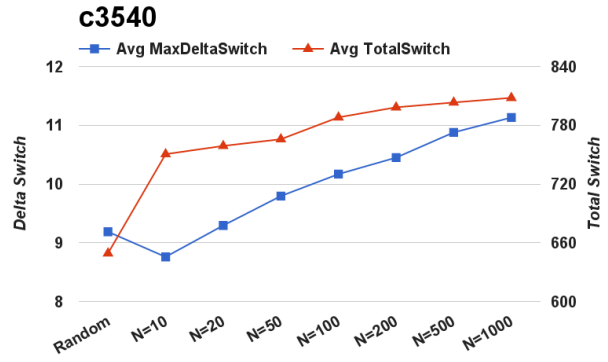
Fig. 6: Impact of N (number of times that a rare node have rare switching) on MaxDeltaSwitch for benchmarks (a) c2670 and (b) c3540.

D. Exploration of N

Fig. 6 shows the distribution of MaxDeltaSwitch over 1000 random 8-trigger Trojan samples for two ISCAS-85 benchmarks. We choose different N to generate MERS testsets, to compare with the Random (10K vectors) testset. For each testset, the box plot shows (minimum, first quartile, median, third quartile, maximum) values of MaxDeltaSwitch of the 1000 Trojan samples. It is clear from these plots that the distribution of MaxDeltaSwitch is constantly improving with increasing N. For $c2670$, the average MaxDeltaSwitch (as shown by the red lines) can reach 18.67 for MERS ($N = 1000$), while Random testset can achieve only 12.15. For $c3540$, the average MaxDeltaSwitch can reach 11.13 for MERS ($N = 1000$), while for Random testset it is only 9.19. The fact that the quality of MERS tests improves with increasing N is not surprising. It is similar to N-detect tests for stuck-at faults, where fault coverage is expected to improve with increasing N. The testset size also increases with N. The sizes of testsets for MERS (N = 10, 20, 50, 100, 200, 500, 1000) are (71, 140, 347, 656, 1262, 3142, 6199) for $c2670$, and (161, 302, 742, 1441, 2858, 7070, 14250) for $c3540$. In most of our experiments, we choose a value of N = 1000, which is a good balance between testset quality and testset size. For fair comparison with Random testset, we will only take the first 10K vectors of MERS testset if it is larger than 10K.



(a)



(b)

Fig. 7: MaxDeltaSwitch versus TotalSwitch for different N for benchmarks (a) c2670 and (b) c3540. MERS creates more switching in Trojan, as well as increased total switching.

E. Effect of Increased Total Switching

Fig. 7 shows the average $MaxDeltaSwitch$ and the average $TotalSwitch$ of the testsets for 1000 8-trigger Trojan samples for different values of N. For both of the two benchmarks, the average $TotalSwitch$ increases with N as well as the average $MaxDeltaSwitch$. It is obvious that all the MERS

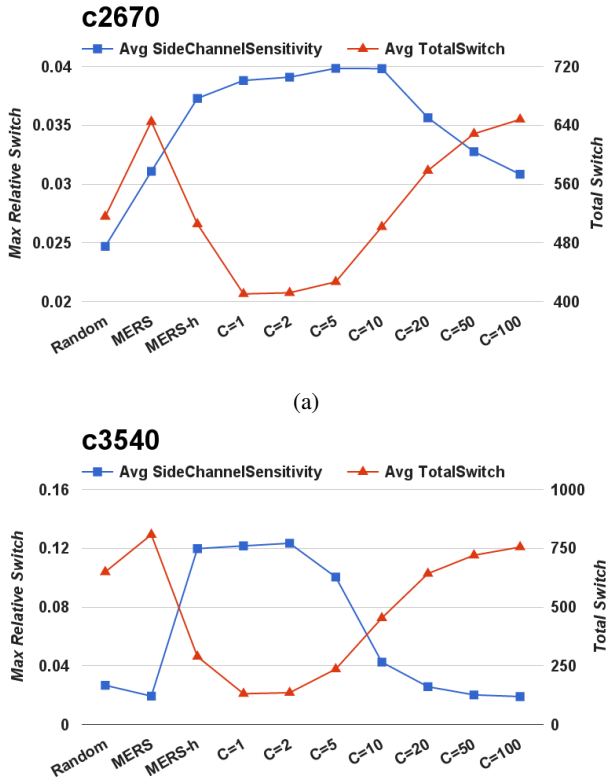


Fig. 8: Side Channel Sensitivity versus $TotalSwitch$ for Random, MERS, MERS-h and MERS-s (with different C) for benchmarks (a) c2670 and (b) c3540.

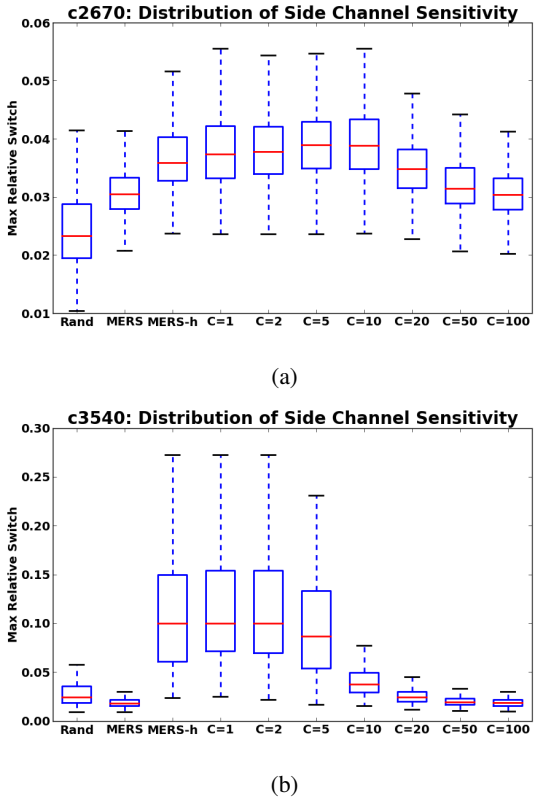


Fig. 9: Distribution of Side Channel Sensitivity for Random, the original MERS, MERS-h and MERS-s (with different C) for benchmarks (a) c2670 and (b) c3540.

testsets have much larger average $TotalSwitch$, compared with the Random testset. For $c2670$, the average $TotalSwitch$ for MERS ($N = 1000$) is 644.9, which is about 1.25X times of that of the Random testset (515.7). For $c3540$, the average $TotalSwitch$ for MERS ($N = 1000$) is 808, while Random testset is only 649.2. The insight that we can get from here is that MERS tends to increase the $TotalSwitch$ of the circuit, although it is designed to increase switches in rare nodes. The following subsection will show that the proposed reordering methods would be effective to reduce $TotalSwitch$ and thus increase side channel sensitivity.

F. Effect of Weight Ratio (C)

The effectiveness of the two reordering methods can be observed in Fig. 8 and Fig. 9. As shown in Fig. 8, MERS-h can reduce $TotalSwitch$ and thus increase the relative switching (i.e. the Side Channel Sensitivity), compared with the original MERS testset. For MERS-s with different weight ratio C, side channel sensitivity improves steadily with a small C, and then goes down when C is too large. As the weight ratio tries to balance $DeltaSwitch$ and $TotalSwitch$, a large C will outweigh the influence of $TotalSwitch$, which will make it less different from the original MERS testset. In the following experiments, we choose the weight ratio as $C = 5$, as it provides a good balance between the total switching and rare switching.

Fig. 9 shows detailed distribution of Side Channel Sensitivity for 1000 8-trigger Trojan samples with different choices of C. The reordering methods are working well to improve Side Channel Sensitivity, which is built on the fact that the original MERS testset is already of high quality in terms of $DeltaSwitch$, or switching in Trojans.

G. Increase in Trojan Activity

Figure 10 shows the distribution of the change in side-channel sensitivity for two benchmarks, comparing with Random testset and MERO [7]. In Figure 10(a) and (b), we can see that our two approaches (MERS-h and MERS-s) can greatly improve the SCS compared with Random testsets as well as MERO [7]. Figure 10(c) and (d) show the Delta SCS when we look at each Trojan. Table II summarizes the percentage of Trojans whose SCS increased with our approaches. Compared with Random testsets, more than 96.2% Trojans (for 1000 samples) have higher SCS with our approaches. Compared with MERO testsets, more than 89.1% Trojans have higher SCS with our approaches.

TABLE II: Percent of Trojans (for 1000 Trojan samples) with SCS increased

Benchmark	MERSh-Rand	MERSs-Rand	MERSh-MERO	MERSs-MERO
c2670	96.2%	97.1%	89.1%	91.2%
c3540	99.8%	99.5%	92.3%	93.4%

Table III shows that MERS (N=1000) is very effective in creating $DeltaSwitch$ caused by arbitrary Trojans due to its

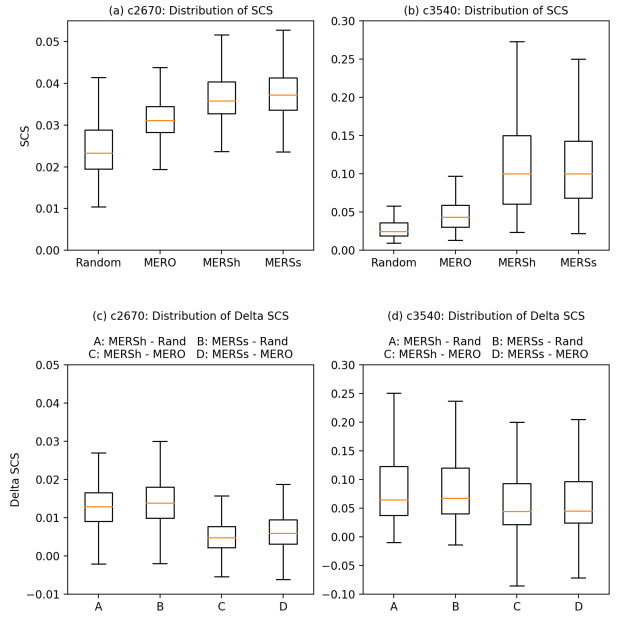


Fig. 10: Distribution of Side Channel Sensitivity (SCS) and Delta SCS (1000 Trojan samples), comparing our approaches with Random testsets and MERO [7].

statistical nature. The average *Max Delta Switch* increases by 31.11% and the average *Avg Delta Switch* by 187.33% on average for different benchmarks, compared with Random testset. This shows the effectiveness of MERS in creating Trojan activity.

Table IV shows that MERS is also helpful in improving RelativeSwitch. The average AvgRelativeSwitch increased by 158.16%, compared with Random testsets. For average MaxRelativeSwitch (Side Channel Sensitivity), MERS has an average improvement of 18.89%. However, Side Channel Sensitivity values for benchmark $c3540$ and $c6288$ are not as good as those of Random testsets. This is due to the fact that MERS testset also increases the total switching, when it is making efforts to cause rare nodes switching. This phenomenon is illustrated and explained in Fig. 7 and Fig. 8, and this side effect can be improved by the two reordering algorithms as shown in Table V and VI.

TABLE III: Comparison of MERS (N=1000) with Random (10K) for average MaxDeltaSwitch and average AvgDeltaSwitch, over 1000 random 8-trigger Trojans.

Benchmark	average MaxDeltaSwitch			average AvgDeltaSwitch		
	Random	MERS	Improv.	Random	MERS	Improv.
c2670	12.15	18.67	53.67%	1.4289	6.8561	379.83%
c3540	9.19	11.13	21.16%	1.3716	2.9058	111.85%
c5315	9.51	13.80	45.16%	1.3116	3.9300	199.64%
c6288	6.63	7.26	9.63%	1.0636	4.8448	355.50%
c7552	8.53	12.00	40.76%	1.3488	2.7700	105.36%
s13207	6.63	8.83	33.18%	0.6428	0.9771	52.01%
s15850	7.53	10.84	43.99%	0.7465	1.3609	82.29%
s35932	15.16	15.37	1.35%	2.1803	6.8060	212.16%
Avg. Improv.	–	–	31.11%	–	–	187.33%

TABLE IV: Comparison of MERS (N=1000) with Random (10K) for average $MaxRelativeSwitch$ (Side Channel Sensitivity) and average $AvgRelativeSwitch$, over 1000 random samples of 8-trigger Trojans.

Benchmark	average MaxRelativeSwitch (Side Channel Sensitivity)			average AvgRelativeSwitch		
	Random	MERS	Improv.	Random	MERS	Improv.
c2670	0.02469	0.03108	25.90%	0.00255	0.01054	314.14%
c3540	0.02670	0.01933	-27.59%	0.00214	0.00361	69.12%
c5315	0.00526	0.00766	45.72%	0.00075	0.00200	165.65%
c6288	0.00534	0.00395	-26.06%	0.00059	0.00219	270.68%
c7552	0.00452	0.00852	88.48%	0.00058	0.00113	94.65%
s13207	0.00756	0.00844	11.64%	0.00066	0.00085	28.22%
s15850	0.00593	0.00716	20.70%	0.00053	0.00082	54.25%
s35932	0.00523	0.00587	12.29%	0.00060	0.00223	268.54%
Avg. Improv.	–	–	18.89%	–	–	158.16%

H. Side Channel Sensitivity Improvement

To this point, we have explored the parameters: N for MERS and C for MERS-s. We choose $N = 1000$ and $C = 5$ in the following experiment to compare our proposed schemes with Random testset and MERO. Table V and VI show the improvement of proposed approaches on Side Channel Sensitivity for 4-trigger and 8-trigger Trojans.

Table V shows that MERS, MERS-h and MERS-s have 10.37%, 138.44% and 152.26% improvement over the Random testsets, respectively. While the original MERS testsets is 23.95% worse than MERO testsets, MERS-h and MERS-s have 52.62% and 62.01% improvement over MERO. Table VI shows the results for 8-trigger Trojans. Compared to Random testsets, MERS, MERS-h and MERS-s can have 18.89%, 107.53% and 96.61% improvement, respectively. The original MERS testsets is 12.43% worse than MERO testsets. MERS-h and MERS-s testsets can improve the Side Channel Sensitivity by 40.79% and 38.50%, respectively.

In this section, we explore the impact of different values of N for MERS and observe the effectiveness of MERS to maximize Trojan activity as N increases. We confirm the superiority of MERS testsets over Random testsets in Section IV-G on creating switching activity in randomly sampled Trojans. We observed that the total switching was also likely to increase while MERS made efforts to maximize rare switching in Trojans. The two reordering methods (MERS-h and MERS-s) successfully had the total switching under control while maintaining the rare switching high.

V. Scalability to Large Designs

In this section, we investigate the scalability of our approach to large designs. We compare the controllability of different scan modes and their effects on side channel sensitivity. We apply region-based MERS on the three sequential ISCAS benchmarks and side channel sensitivity can improve as we divide the design into more regions. We also generate tests for two large benchmarks (AES cipher and DLX processor) from OpenCores and design partitioning can significantly improve the side channel sensitivity.

A. Controllability of Different Scan Modes

The scan modes have direct influence on the effectiveness of our region-based MERS approach. Fig. 11 shows the *Total*

TABLE V: Comparison of average **Side Channel Sensitivity** between Random (10K), MERO, and MERS testsets, N=1000, C=5 for MERS-s, over 1000 random samples of 4-trigger Trojans.

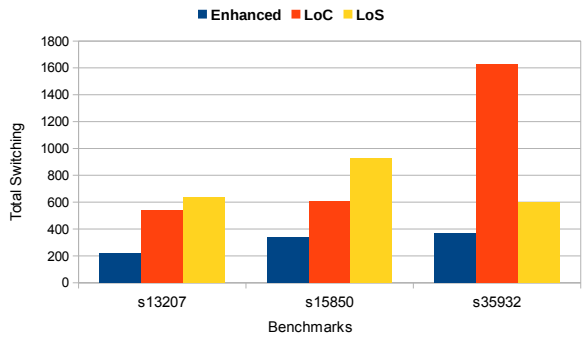
Benchmark	Comparison Testsets		Proposed Schemes			Improvement to Random			Improvement to MERO		
	Random	MERO	MERS	MERS-h	MERS-s	MERS	MERS-h	MERS-s	MERS	MERS-h	MERS-s
c2670	0.01703	0.02571	0.02231	0.03035	0.03308	31.01%	78.27%	94.31%	-13.23%	18.07%	28.69%
c3540	0.02144	0.04238	0.01336	0.10677	0.11067	-37.71%	397.97%	416.16%	-68.48%	151.96%	161.16%
c5315	0.00445	0.01082	0.00747	0.01287	0.01586	67.79%	188.97%	256.29%	-30.97%	18.89%	46.59%
c6288	0.00480	0.00395	0.00313	0.00741	0.00896	-34.81%	54.47%	86.85%	-20.88%	87.50%	126.80%
c7552	0.00351	0.00737	0.00491	0.01250	0.01168	39.61%	255.63%	232.38%	-33.46%	69.50%	58.42%
s13207	0.00568	0.00617	0.00619	0.00773	0.00826	9.07%	36.24%	45.49%	0.31%	25.29%	33.80%
s15850	0.00447	0.00487	0.00474	0.00691	0.00634	6.14%	54.83%	42.06%	-2.75%	41.86%	30.17%
s35932	0.00354	0.00463	0.00361	0.00500	0.00512	1.89%	41.17%	44.53%	-22.12%	7.90%	10.48%
Avg. Improve.	–	–	–	–	–	10.37%	138.44%	152.26%	-23.95%	52.62%	62.01%

TABLE VI: Comparison of average **Side Channel Sensitivity** between Random (10K), MERO, and MERS testsets, N=1000, C=5 for MERS-s, over 1000 random samples of 8-trigger Trojans.

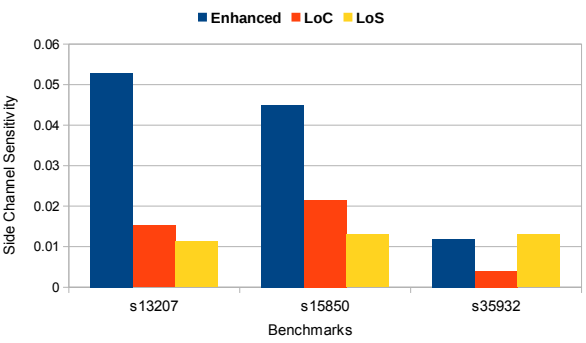
Benchmark	Comparison testsets		Proposed Schemes			Improvement to Random			Improvement to MERO		
	Random	MERO	MERS	MERS-h	MERS-s	MERS	MERS-h	MERS-s	MERS	MERS-h	MERS-s
c2670	0.02469	0.03204	0.03108	0.03729	0.03984	25.90%	51.05%	61.40%	-3.01%	16.37%	24.35%
c3540	0.02670	0.05532	0.01933	0.11974	0.10037	-27.59%	348.53%	275.96%	-65.05%	116.47%	81.44%
c5315	0.00526	0.00875	0.00766	0.01020	0.01129	45.72%	94.03%	114.78%	-12.38%	16.66%	29.14%
c6288	0.00534	0.00412	0.00395	0.00649	0.00790	-26.06%	21.55%	47.97%	-4.20%	57.49%	91.72%
c7552	0.00452	0.00914	0.00852	0.01437	0.01149	88.48%	217.78%	154.00%	-6.70%	57.31%	25.74%
s13207	0.00756	0.00838	0.00844	0.01053	0.01112	11.64%	39.24%	47.05%	0.69%	25.58%	32.63%
s15850	0.00593	0.00722	0.00716	0.00923	0.00818	20.70%	55.69%	37.94%	-0.87%	27.86%	13.28%
s35932	0.00523	0.00638	0.00587	0.00692	0.00700	12.29%	32.39%	33.80%	-7.90%	8.58%	9.74%
Avg. Improve.	–	–	–	–	–	18.89%	107.53%	96.61%	-12.43%	40.79%	38.50%

Switching and the *Side Channel Sensitivity* when different scan modes are used for region-based MERS approach (each benchmark has four regions in this example). In Fig. 11(a), the Enhanced mode can greatly reduce the *Total Switching* compared with LoC and LoS. In Fig. 11(b), the Enhanced mode can greatly improve the *Side Channel Sensitivity* compared with LoC and LoS.

There are two factors that enable the *Enhanced* mode to do much better than the LoC and LoS modes. (1) We try to reduce the background switching by assigning 0's to the flip-flops that are outside of the targeted region. For the *Enhanced* mode, we assign 0's to both V_1 and V_2 for those flip-flops outside of the targeted region. This enables the *Enhanced* mode to have full controllability to turn the other regions "dark". For the LoC and LoS modes, we assign 0's to V_1 for those flip-flops that are outside of the targeted region, while V_2 cannot directly assign values to flip-flops. For LoC mode, the flip-flop states before capture will be the states after the circuit propagates one cycle after V_1. For LoS mode, the flip-flop states will be shifted by 1 from V_1. (2) The *Enhanced* mode has the benefit of using V_2 to assign arbitrary values to the flip-flops inside the targeted region. In contrast, LoC has no direct control over the states after one clock cycle and LoS has to assign the in-region flip-flops to V1 shifted by 1. In our MERS approach (Algorithm 2), we mutate both the PIs and the pseudo-PIs (i.e. the values for the in-region flip-flops) to generate high quality test for each region. Under the *Enhanced* mode, we can mutate the vector V_2 to find beneficiary values for pseudo-PIs.



(a) Total Switching



(b) Side Channel Sensitivity

Fig. 11: Comparison of Enhanced, LoC and LoS modes.

B. Effectiveness of Design Partitioning

Fig. 12 shows the results for region-based MERS approach on the three sequential benchmarks. We compare the MERS testsets produced by 1 region, 2 regions, 4 regions and 8 regions. The *Total Switching* and *Side Channel Sensitivity* numbers are the averaged values of 1000 random in-region (out-region) Trojans. Here an in-region Trojan means that its trigger edges and payload edge belong to the same region. We have 125 random in-region Trojans from each of the 8 regions

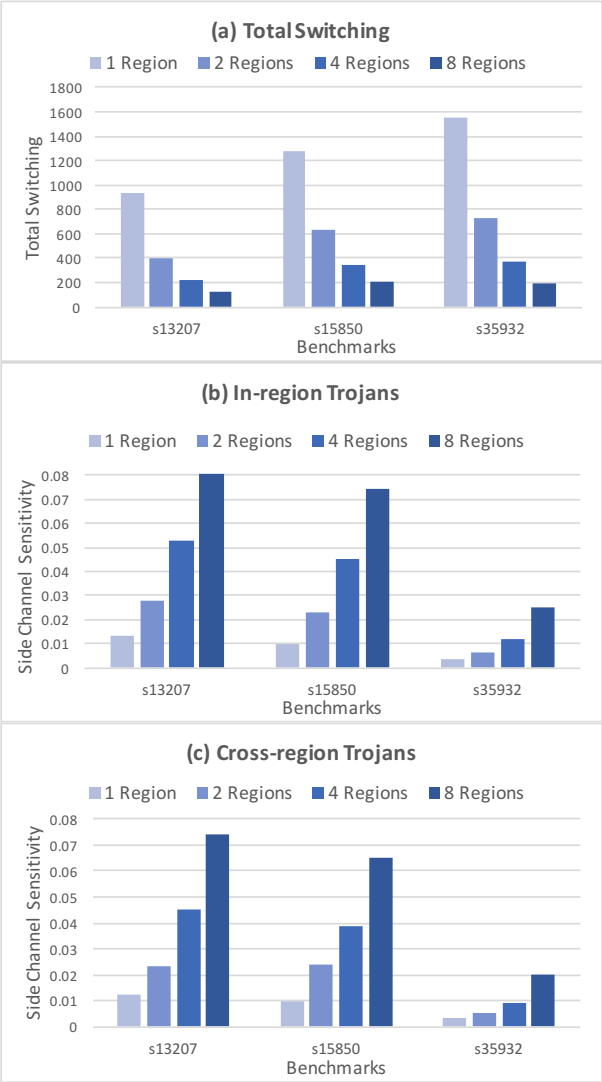


Fig. 12: (a) *Total Switching*. (b) *Side Channel Sensitivity* for in-region Trojans samples. (c) *Side Channel Sensitivity* for cross-region Trojans.

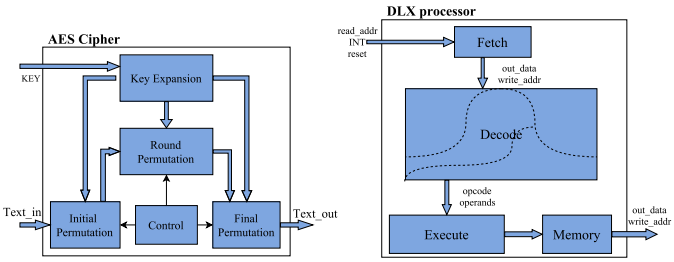
to form a set of 1000 random Trojans for each benchmark.

As shown in Fig. 12(a), the averaged *Total Switching* decreases drastically as we partition the design into more regions. As the number of regions doubles, the averaged *Total Switching* reduces almost by half. MERS with 8 regions can reduce the *Total Switching* by 7.40X for s13207, 6.28X for s15850, and 8.12X for s35932, compared to MERS with only 1 region. As shown in Fig. 12(b), *Side Channel Sensitivity* improves significantly as the number of regions increases for in-region Trojans. MERS with 8 regions can improve the *Side Channel Sensitivity* by 6.24X for s13207, 7.51X for s15850, and 7.49X for s35932, compared to MERS with only 1 region.

Fig. 12(c) shows the average SCS of 1000 cross-region Trojan samples. We can still see the trend that SCS will greatly increase as we divide the design into more regions. We observe slightly lower SCS compared with in-region Trojan samples. For the benchmark s13207, cross-region Trojan samples have 16.1% less SCS for *2 Regions*, 14.0% less SCS for *4 Regions*, and 7.9% less SCS for *8 Regions*, compared with in-region Trojan samples. For the benchmark s35932, cross-region Trojan samples have 18.8% less SCS for *2 Regions*, 22.5% less SCS for *4 Regions*, and 20.3% less SCS for *8 Regions*, compared with in-region Trojan samples. Thus the conclusion is that cross-region Trojans can still significantly benefit from the proposed approach.

C. Test Generation for Large OpenCores Benchmarks

In this subsection, we apply our region-based MERS approach on two large designs (AES cipher and DLX processor). AES cipher has 15086 nodes and DLX processor has 18123 nodes. They are about three times as large as the largest ISCAS benchmark s35932. The results show that our approach is scalable for large designs. Direct application of MERS on AES takes about 7 days to generate and reorder tests, and about 9 days for DLX. After functionally partitioning AES into three regions, the largest region can finish in 4 days (we generate the tests for each region in parallel). After functional partitioning of DLX and structural partitioning of its decode module, we can finish the test generation and reordering for DLX in 3 days. In this part, we assume that the designs are equipped with enhanced scan chain, which provides us the most controllability for test generation.



(a) Functional partition on AES (b) Functional partition on DLX + structural partition on Decode.

Fig. 13: Design partition for AES and DLX.

1) **AES:** Fig. 13a shows the abstracted representation of an AES cipher. We use functional partition to segment it into three regions. It has two obvious submodules: *Key Expansion* and *Round Permutation*, which we choose as two regions. The third region contains the rest of the circuit, which is mostly the control logic and input/output buffers.

Table VII compares the side channel sensitivity on 15 random Trojans for three testsets: Random, MERS (whole design), MERS-FP (with functional partition). For the MERS and MERS-FP, we use the test generation detailed in Algorithm 2 and the test reordering detailed in Algorithm 3. Compared with the Random testset, the MERS testset can improve the side channel sensitivity by only 5% on average. The MERS-FP testset can improve the side channel sensitivity by 160% on average. The functional partition significantly improved the side channel sensitivity (about 2.6X) over the Random testset.

2) **DLX:** Fig. 13b shows the abstracted representation of a DLX processor. We use functional partition to segment it into four regions: *Fetch*, *Decode*, *Execute*, and *Memory*. However, the Decode consumes majority of the chip area (accounting

TABLE VII: AES testsets: [Random, MERS, MERS-FP]

Testset	Random	MERS	MERS-FP
Trojan 1	0.00125	0.00154	0.00568
Trojan 2	0.00119	0.00135	0.00538
Trojan 3	0.00136	0.00143	0.00525
Trojan 4	0.00170	0.00151	0.00573
Trojan 5	0.00125	0.00167	0.00568
Trojan 6	0.00107	0.00132	0.00196
Trojan 7	0.00105	0.00143	0.00235
Trojan 8	0.00148	0.00158	0.00196
Trojan 9	0.00090	0.00083	0.00110
Trojan 10	0.00083	0.00107	0.00127
Trojan 11	0.00267	0.00237	0.00389
Trojan 12	0.00227	0.00255	0.00356
Trojan 13	0.00235	0.00226	0.00952
Trojan 14	0.00262	0.00235	0.00635
Trojan 15	0.00235	0.00228	0.00354
Average SCS	0.00162	0.00170	0.00421
Average Improve.	-	5%	160%

TABLE VIII: DLX testsets: [Random, MERS, MERS-FP, MERS-FP+SP]

Testset	Random	MERS	MERS-FP	MERS-FP+SP
Trojan 1	0.00045	0.00059	0.00059	0.00202
Trojan 2	0.00055	0.00067	0.00067	0.01515
Trojan 3	0.00066	0.00087	0.00087	0.00448
Trojan 4	0.00050	0.00062	0.00065	0.00448
Trojan 5	0.00079	0.00059	0.00059	0.00202
Trojan 6	0.00042	0.00085	0.00085	0.00285
Trojan 7	0.00061	0.00090	0.00090	0.00384
Trojan 8	0.00081	0.00099	0.00099	0.00632
Trojan 9	0.00051	0.00096	0.00096	0.00210
Trojan 10	0.00059	0.00086	0.00126	0.00673
Trojan 11	0.00045	0.00073	0.00078	0.02273
Trojan 12	0.00038	0.00067	0.00067	0.00202
Trojan 13	0.00075	0.00093	0.00093	0.00384
Trojan 14	0.00051	0.00079	0.00079	0.00210
Trojan 15	0.00050	0.00062	0.00062	0.01515
Average SCS	0.00056	0.00077	0.00081	0.00639
Average Improve.	-	37%	43%	1033%

for 71% of the whole design area). We used the hypergraph partitioning tool hMETIS [27] to further partition the Decode region into four regions of roughly equal size (in terms of number of gates/vertices).

Table VIII compares the side channel sensitivity on 15 random Trojans for four testsets: Random, MERS (whole design), MERS-FP (with functional partition), MERS-FP+SP (with functional partition followed by structural partition). Compared with the Random testset, the MERS testset can improve the side channel sensitivity by about 37% on average, and MERS-FP testset can improve by 43%. The MERS and MERS-FP has very close side channel sensitivity numbers, because the Decode region is very huge. The MERS-FP+SP testset can significantly improved the side channel sensitivity (about 11X times) over the Random testset. The experiments on AES and DLX have shown that our approach is scalable to large designs to greatly improve the side channel sensitivity for hardware Trojan detection.

VI. Conclusions

We have presented a framework for scalable test generation, called MERS, which can significantly improve the Trojan detection sensitivity in side-channel analysis based Trojan detection. The approach aims at statistically increasing switching activity in an unknown Trojan to amplify the Trojan effect in presence of large process variations. Such a test generation approach will, in general, be effective for any side-channel analysis approaches that rely on activity in Trojan circuits (e.g. transient current, dynamic power profile, or electromagnetic emanation based methods). MERS is effective for any Trojan forms/sizes, as long as a Trojan is implanted through alterations in a circuit structure - the most dominant mode of Trojan implantation. Our simulation results on a set of benchmark circuits show that the proposed approach can significantly improve the side channel sensitivity by 97%, compared with random tests for a large set of arbitrary Trojans. Furthermore, our approach is scalable to large designs (e.g. AES cipher and DLX processor), which can improve side channel sensitivity by 1.6X times for AES, and 10X times for DLX. Further, the approach can work for different DFT configurations. Our results demonstrated that a scalable statistical test generation can serve as an essential component in any side channel analysis based hardware Trojan detection framework.

VII. Acknowledgements

This work was partially supported by grants from National Science Foundation (1441667, 1603475, 1603483), Semiconductor Research Corporation (2014-TS-2554) and Cisco Systems (F020375). Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] Y. Huang, S. Bhunia, P. Mishra. MERS: Statistical Test Generation for Side-Channel Analysis based Trojan Detection. ACM Conference on Computer and Communications Security (CCS), pp. 130-141, 2016.

[2] P. Mishra, S. Bhunia and M. Tehranipoor (Editors). Hardware IP Security and Trust, ISBN 9783319490250, Springer 2017.

[3] DARPA: TRUST in Integrated Circuits (TIC), 2007. [Online]. Available: http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA503809

[4] R. Chakraborty and S. Bhunia. Security against hardware Trojan through a novel application of design obfuscation. ACM International Conference on Computer-Aided Design (ICCAD), pp. 113-116, 2009.

[5] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi and V. De. Parameter variations and impact on circuits and microarchitecture. ACM/IEEE Design Automation Conference (DAC), pp. 338-342, 2003.

[6] F. Wolff, C. Papachristou, S. Bhunia and R. S. Chakraborty. Towards Trojan-Free Trusted ICs: Problem Analysis and Detection Scheme. Design, Automation and Test in Europe (DATE), pp. 1362-1365, 2008.

[7] R. Chakraborty, F. Wolff, S. Paul, C. Papachristou and S. Bhunia. MERO: A Statistical Approach for Hardware Trojan Detection. International Workshop on Cryptographic Hardware and Embedded Systems, pp. 396-410, 2009.

[8] S. Saha, R. Chakraborty, S. Nuthakki, Anshul, and D. Mukhopadhyay. Improved Test Pattern Generation for Hardware Trojan Detection Using Genetic Algorithm and Boolean Satisfiability. International Workshop on Cryptographic Hardware and Embedded Systems, pp. 577-596, 2015.

[9] M. Banga and M. Hsiao. A region based approach for the identification of hardware Trojans. IEEE International Workshop on Hardware-Oriented Security and Trust (HOST), 2008.

[10] M. Banga, M. Chandrasekar, L. Fang and M. Hsiao. Guided test generation for isolation and detection of embedded Trojans in ICs. ACM Great Lakes Symposium on VLSI (GLSVLSI), pp. 363-366, 2008.

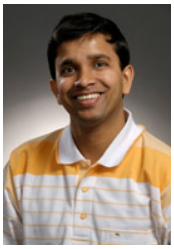
[11] Y. Jin and Y. Makris. Hardware Trojan detection using path delay fingerprint. IEEE International Workshop on Hardware-Oriented Security and Trust (HOST), 2008.

[12] S. Wei and M. Potkonjak. Scalable hardware Trojan diagnosis. IEEE Transactions on Very Large Scale Integration Systems (TVLSI), 20(6), pp. 1049-1057, 2012.

[13] R. Rad, J. Plusquellic and M. Tehranipoor. A sensitivity analysis of power signal methods for detecting hardware Trojans under real process and environmental conditions. IEEE Transactions on Very Large Scale Integration Systems (TVLSI), 18(12), pp. 1735-1744, 2010.
[14] H. Salmani and M. Tehranipoor. Layout-Aware Switching Activity Localization to Enhance Hardware Trojan Detection. IEEE Transactions on Information Forensics and Security (TIFS), 7(1), pp. 76-87, 2012.
[15] S. Dupuis, P. Ba, G. Natale, M. Flottes, and B. Rouzeyre. A novel hardware logic encryption technique for thwarting illegal overproduction and Hardware Trojans. IEEE 20th International On-Line Testing Symposium (IOLTS), pp. 49-54, 2014.
[16] J. Rajendran, Y. Pino, O. Sinanoglu and R. Karri. Security analysis of logic obfuscation. ACM/IEEE Design Automation Conference (DAC), pp. 83-89, 2012.
[17] S. Shekarian, M. Zamani and S. Alami. Neutralizing a design-for-hardware trust technique. International Symposium on Computer Architecture and Digital Systems (CADS), pp. 73-78, 2013.
[18] X. Mingfu, H. Aiqun and L. Guyue: Detecting Hardware Trojan through Heuristic Partition and Activity Driven Test Pattern Generation. Communications Security Conference (CSC), pp. 1-6, 2014.
[19] Y. Huang and P. Mishra. Trace Buffer Attack on the AES Cipher. Springer Journal of Hardware and Systems Security (HASS), 1(1), pages 68-84, 2017.
[20] F. Farahmandi, Y. Huang and P. Mishra. Trojan Localization using Symbolic Algebra. Asia and South Pacific Design Automation Conference (ASPDAC), pages 591-597, 2017.
[21] H. Salmani, M. Tehranipoor and J. Plusquellic. A novel technique for improving hardware Trojan detection and reducing Trojan activation time. IEEE Transactions on Very Large Scale Integration Systems (TVLSI), 20(1), pp. 112-125, 2012.
[22] B. Zhou, W. Zhang, S. Thambipillai, and J. Teo. A low cost acceleration method for hardware Trojan detection based on fan-out cone analysis. ACM International Conference on Hardware Software Codesign and System Synthesis, p. 28, 2014.
[23] I. Pomeranz and S. Reddy. A measure of quality for n-detection test sets. IEEE Transactions on Computers, 53(11), pp. 1497-1503, 2004.
[24] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi and B. Sunar. Trojan Detection using IC Fingerprinting. IEEE Symposium on Security and Privacy, pp. 296-310, 2007.
[25] X. Wang, H. Salmani, M. Tehranipoor and J. Plusquellic. Hardware Trojan Detection and Isolation Using Current Integration and Localized Current Analysis. IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems, pp. 87-95, 2008.
[26] D. Du, S. Narasimhan, R. Chakraborty and S. Bhunia. Self-referencing: a scalable side-channel approach for hardware trojan detection. International Workshop on Cryptographic Hardware and Embedded Systems (CHES), pp. 173-187, 2010.
[27] Karypis, G., Aggarwal, R., Kumar, V. and Shekhar, S. Multilevel hypergraph partitioning: applications in VLSI domain. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 7(1), pp.69-79, 1999.
[28] I. Park and E. J. McCluskey. Launch-on-Shift-Capture Transition Tests, IEEE International Test Conference, Santa Clara, CA, 2008, pp. 1-9.
[29] A. K. Suhag and V. Shrivastava. Delay Testable Enhanced Scan Flip-Flop: DFT for High Fault Coverage. International Symposium on Electronic System Design, Kochi, Kerala, 2011, pp. 129-133.
[30] Gefu Xu and A. D. Singh. Low Cost Launch-on-Shift Delay Test with Slow Scan Enable. IEEE European Test Symposium (ETS'06), Southampton, 2006, pp. 9-14.
[31] OpenCores, Project *aes_core* and *dlx*. http://www.opencores.org
[32] J. Cruz, Y. Huang, P. Mishra and S. Bhunia. An Automated Configurable Trojan Insertion Framework for Dynamic Trust Benchmarks. Design Automation and Test in Europe (DATE), pp -, Dresden, Germany, March 19 - 23, 2018.
[33] P. Subramanyan, S. Ray and S. Malik. Evaluating the security of logic encryption algorithms. IEEE International Symposium on Hardware Oriented Security and Trust (HOST), 2015, pp. 137-143.

Yuanwen Huang received the B.E. degree from Huazhong University of Science and Technology, China, in 2012. He received the Ph.D. degree in Computer Engineering from University of Florida, in 2017. He was a recipient of the Best Paper Award from the International Symposium on Quality Electronic Design in 2016. He is currently a software engineer at VMware. His research interests include energy and reliability optimization in embedded systems, hardware security and trust for integrated circuits.

Swarup Bhunia (Senior Member, IEEE) received his B.E. (Hons.) from Jadavpur University, Kolkata, India, M.Tech. from the Indian Institute of Technology (IIT), Kharagpur, and Ph.D. from Purdue University. Currently, Dr. Bhunia is a professor and Steven Yatauro Faculty Fellow in the University of Florida. Earlier he was appointed as the T. and A. Schroeder associate professor of Electrical Engineering and Computer Science at Case Western Reserve University. He has over 200 publications in peer-reviewed journals and premier conferences. His research interests include hardware security and trust, adaptive nanocomputing and novel test methodologies. Dr. Bhunia received IBM Faculty Award (2013), National Science Foundation career development award (2011), Semiconductor Research Corporation Inventor Recognition Award (2009), and SRC technical excellence award (2005), and several best paper awards/nominations. He has been serving as an associate editor of IEEE Transactions on CAD, IEEE Transactions on Multi-Scale Computing Systems, ACM Journal of Emerging Technologies, and Journal of Low Power Electronics; served as guest editor of IEEE Design & Test of Computers (2010, 2013) and IEEE Journal on Emerging and Selected Topics in Circuits and Systems (2014). He has served as co-program chair of IEEE IMS3TW 2011, IEEE NANOARCH 2013, IEEE VDAT 2014, and IEEE HOST 2015, and in the program committee of many IEEE/ACM conferences. He is a senior member of IEEE.



Prabhat Mishra (S00-M04-SM08) is a Professor in the Department of Computer and Information Science and Engineering at the University of Florida. His research interests include embedded and cyber-physical systems, energy-aware computing, hardware security and trust, system-on-chip verification, bioinformatics, and post-silicon debug. He received his Ph.D. in Computer Science and Engineering from the University of California, Irvine. He has published six books and more than 150 research articles in premier international journals and conferences. His research has been recognized by several awards including the NSF CAREER Award, IBM Faculty Award, three best paper awards, and EDAA Outstanding Dissertation Award. Prof. Mishra currently serves as the Deputy Editor-in-Chief of IET Computers & Digital Techniques, and as an Associate Editor of ACM Transactions on Design Automation of Electronic Systems, IEEE Transactions on VLSI Systems, and Journal of Electronic Testing. Prof. Mishra is an ACM Distinguished Scientist and a Senior Member of IEEE.