Automated Test Generation for Trojan Detection using Delay-based Side Channel Analysis

Yangdi Lyu and Prabhat Mishra
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, Florida, USA

***Abstract*—Side-channel analysis is widely used for hardware Trojan detection in integrated circuits by analyzing various side-channel signatures, such as timing, power and path delay. Existing delay-based side-channel analysis techniques have two major bottlenecks: (i) they are not suitable in detecting Trojans since the delay difference between the golden design and a Trojan inserted design is negligible, and (ii) they are not effective in creating robust delay signatures due to reliance on random and ATPG based test patterns. In this paper, we propose an efficient test generation technique to detect Trojans using delay-based side channel analysis. This paper makes two important contributions. (1) We propose an automated test generation algorithm to produce test patterns that are likely to activate trigger conditions, and change critical paths. Compared to existing approaches where delay difference is solely based on extra gates from a small Trojan, the change of critical paths by our approach will lead to significant difference in path delay. (2) We propose a fast and efficient reordering technique to maximize the delay deviation between the golden design and Trojan inserted design. Experimental results demonstrate that our approach significantly outperforms state-of-the-art approaches that rely on ATPG or random test patterns for delay-based side-channel analysis.**

***Index Terms*—Trojan detection, path delay, test generation**

I. Introduction

Malicious implants are widely acknowledged as a major threat in System-on-Chip (SoC) design methodology due to the inherent supply chain vulnerabilities [1], [2]. Hardware Trojans can be inserted by malicious third party to either alter the functionality or leak information from the design. In order to design trustworthy SoCs, it is critical to ensure that the IPs manufactured by third parties are Trojan-free. Detection of hardware Trojans is challenging due to their stealthy nature. A Trojan normally consists of a rare trigger condition and a payload. Trigger condition is carefully crafted such that it is only activated under extremely rare conditions. The functionality of a design remains exactly the same as the golden design when the trigger condition is not satisfied. An example Trojan is shown in Figure 1, where $asset$ represents the signal that the attacker wants to invert by creating a trigger using four rare signals, x_1, x_2, x_3, x_4.

Side-channel analysis is promising for Trojan detection, but it faces two major challenges. The first challenge comes from the process variation and environmental noise. As transistor dimensions continue to shrink, it introduces increasing process variations across integrated circuits (ICs) of the same design. Since the Trojans are typically small (e.g., few gates in a million-gate IC), the deviation due to the presence of a Trojan is negligible with respect to process variation and environmental noise. As a result, any measured deviation in side-channel signature cannot guarantee the existence of a Trojan. The second challenge is how to automatically generate high quality test patterns that can sensitize critical paths. The state-of-the-art path delay based approaches utilize random or ATPG-based test generation techniques. However, the delay difference generated by these approaches is typically too small to provide high confidence in Trojan detection.

This work was partially supported by the NSF grant CCF-1908131.

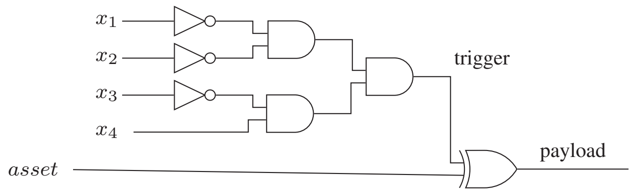

Fig. 1. An example Trojan with a trigger consisting of four rare signals. When the trigger is activated, the payload gets the inverted *asset* value.

To overcome these challenges, we propose an automated approach to generate high quality test patterns for path delay based side-channel analysis to significantly improve the side-channel sensitivity. The main observation is that the tests generated by logic testing are more likely to activate trigger conditions, and by utilizing these tests, we can produce two completely different critical paths for the same register in the golden design and in a Trojan-inserted design. As a result, it can lead to significantly different path delays, compared to the negligible delay introduced by few extra gates (from a Trojan) in a fixed critical path. In this paper, we make the following three important contributions:

1) We propose an efficient test generation method to maximize observable path delays by changing critical paths.
2) We design a lightweight and effective logic testing algorithm to generate tests for delay-based side-channel analysis. The generated tests assume no preliminary information about critical paths or trigger conditions.
3) We perform a Hamming-distance based reordering of the generated tests. We design a distance evaluation method to increase the probability of constructing a critical path from the trigger to the payload.

The remainder of the paper is organized as follows. We present the related work in Section II. Section III describes our automated test generation approach. Section IV presents the experimental results. Finally, Section V concludes the paper.

II. Background and Related Work

A. Hardware Trojan Detection

There are a wide variety of approaches for Trojan detection. Logic testing compares the outputs of an implementation to a golden specification [3]–[23]. Side-channel analysis, on the other hand, examines side effects of the inserted Trojans, such as power, dynamic current, and path delay [24]–[33]. Given the exponential test space complexity, it may not be feasible to activate a rare trigger condition and propagate the Trojan effects to observable outputs. Probabilistic test generation techniques are promising (e.g., N-detect approach [3]) to increase the likelihood of Trojan detection. While side-channel analysis does not have these requirements, it has a major challenge of low side-channel sensitivity [30]. Since the inserted Trojan is relatively small (few gates) in a large design, the side-channel footprint of a Trojan is likely to be dominated by process variations and environmental noise margins.

B. Path Delay based Side-channel Analysis

Path delay based side-channel analysis is beneficial compared to other side-channel parameters as the delay of each output can be measured independently, and an inserted Trojan may affect multiple observable outputs. The delay is expected to be greater than the delay in the golden design with extra gates inserted. The main challenge in path delay based approach is to find suitable input pattern (test) that can reveal delay difference introduced by the Trojan. Existing approaches apply a static analysis on the design to find all possible paths, and use Automatic Test Pattern Generation (ATPG) tools to generate test patterns that are able to sensitize these paths. For example, Jin et al. [25] used Synopsys TetraMAX to analyze the design and generate test patterns to cover every path. However, this approach is time-consuming and not scalable for large designs since the number of possible paths grows exponentially with the size of the design. In addition, the small delay difference introduced by Trojan is likely to be dominated by large process variation and environmental noise. In this paper, we propose an approach to significantly increase the delay difference by changing the critical paths to offset possible noise. Note that existing statistical and learning approaches are applicable to the delay profiles generated by our approach to achieve better results.

C. Trojan Effects on Path Delay

There are two types of effects that a Trojan has over path delay. The first one comes from the change of fan-out. In Figure 1, as the trigger points $(x_1, ..., x_4)$ connect to an extra gate compared to the golden design, the gates that produce these signals will change their capacitive load. As a result, the propagation delay of these gates will increase. The other type of impact is from the extra gates that are inserted by the payload. For example, the XOR gate in Figure 1 is inserted to change the value of *asset* when the trigger is activated. This extra XOR gate adds to the total path delay for any path passing through it.

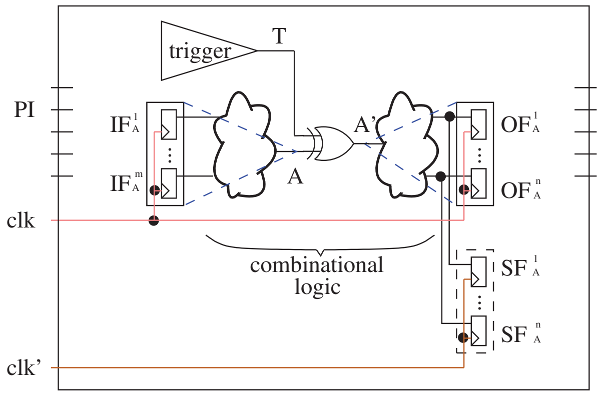

Fig. 2. Path delay measurement using shadow registers [34]. The registers of the input layer and the output layer of the asset A are represented using IF_A^i and OF_A^j, respectively. The path delay is measured by tuning clk' and comparing the values in the corresponding register of the output layer and the shadow registers, e.g., OF_A^1 and SF_A^1.

III. Test Generation for Path Delay Analysis

The main challenge in Trojan detection using delay-based side-channel analysis is how to increase the observability. One of the common methods to measure path delay is using shadow registers [34]. As shown in Figure 2, the original registers and shadow registers utilize different clocks to measure delays by controlling the skew of *clk* and *clk'*. The original *clk* is used to maintain the correct functionality, while the second *clk'* can be tuned to find out the exact time of a signal flipping by comparing the values in corresponding registers. As a result, there would be no delay if the signal value does not change between two simulations. For example, when the value of OF_A^1 remains the same between two simulations, SF_A^1 will have the same value as OF_A^1 irrespective of how clk' is tuned, thus, no delay information can be retrieved.

To observe the delay caused by the inserted Trojan, the critical path of some register in the output layer[1] of A, e.g., OF_A^1 in Figure 2, needs to contain A'. Otherwise, the delay between the input layer[2] and the output layer will be almost the same between the golden design and Trojan-inserted design (only differed slightly due to capacitance change). With the critical path crossing A', the signal value of A' has to switch to reveal delay information, either by trigger T or by asset A. In addition, there must exist a path from A' to the output layer where all signals need to switch. Our goal is to generate test vectors that are able to maximize the delay difference of a critical path from the Trojan to the output layer.

A. Test Generation for Path Delay Maximization

The activation of a trigger is important in maximizing the delay difference. Existing approaches try to find critical paths that are affected by the Trojan. However, without the activation of a trigger, the delay difference is at most one gate difference. As shown in Figure 3, the trigger signal T remains zero and the

[1]The **output layer** of a signal contains all the registers encountered in the immediately succeeding layer in the path from the signal to primary outputs.

[2]The **input layer** of a signal contains all the registers encountered in the immediately preceding layer in the path from the signal to the primary inputs.

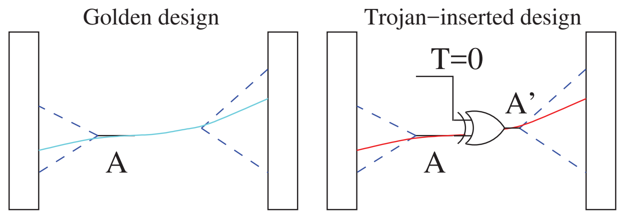

Fig. 3. The delay differs by one gate along the same critical path between the golden design and the Trojan-inserted design in existing approaches.

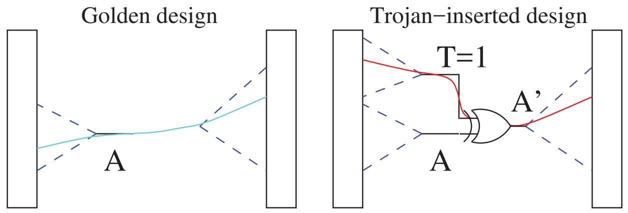

Fig. 4. Our approach maximizes delay difference by changing critical paths.

Trojan-inserted design behaves exactly the same as the original design. As a result, any delay information from the input layer to T is hidden and the delay of A' is determined by A. Assume that we are able to construct a critical path from A to the output layer using a specific test vector. Since the behaviors of the golden design and the Trojan-inserted design are the same, two critical paths are the same except for the extra XOR gate. On the other hand, the critical paths can change significantly when the trigger is activated. Figure 4 shows the optimal scenario of maximizing the delay difference. In Figure 4, the critical path in the Trojan-inserted design goes through the trigger T and propagate the delay to the output layer, which is completely different from the path in the golden design. As a result of two totally different critical paths, the measured delay difference in the output layer can be significantly larger, compared to the scenario when the trigger is not activated in Figure 3.

Therefore, the goal of our test generation technique is to increase the probability of activating trigger conditions. As the attackers are more likely to construct trigger conditions using rare signals, we propose to use a SAT-based approach to generate test patterns in Algorithm 1. It first parses the circuit and computes logic expressions for all rare signals. Then, it repeats k times to generate k test vectors, where k is defined by the user to balance debug time and performance. In the i^{th} iteration, we first randomize the order of rare nodes such that each time the generated tests can cover different sets of rare nodes. Next, we keep adding rare nodes into current trigger CT if CT is still valid. Finally, we use a SAT solver to return a test for CT. Intuitively, we want to generate a test that is able to activate as many rare nodes as possible. Since an adversary wants to hide from side-channel analysis, i.e., introduce the minimum delay, the number of trigger points is typically small. The test that is able to activate many rare nodes has the high probability of covering an unknown trigger condition. Note that the goal of our test generation partially overlaps with logic testing, without the requirement of propagating the effects of payload to the primary output. Experiments show that our lightweight algorithm is effective in delay-based side-channel analysis. Our framework is expected to perform better in the presence of advanced logic testing techniques.

Algorithm 1: Test Generation

```
Input  : circuit netlist, a set of rare nodes ($R$), the
         number of test vectors $k$
Output: test vectors $T = \{t_1, t_2, ..., t_k\}$
1  Parse circuit netlist, and compute logic expression for
     each rare node;
2  Initialize $T = \{\}$;
3  $i = 1$;
4  repeat
5      Current trigger $CT = \emptyset$;
6      Randomize the order of rare nodes $R$;
7      for rare node $r \in R$ do
8          if $CT \cup r$ is a valid trigger then
9              $CT = CT \cup r$;
10         end
11     end
12     Solve $CT$ and get a test $t$;
13     $t_i = t$;
14     $i = i + 1$;
15 until $i > k$;
16 Return $T = \{t_1, t_2, ..., t_k\}$
```

B. Hamming-distance based Reordering

Activating the trigger is not a sufficient condition to introduce delay of the Trojan to the output layer. It also requires construction of a critical path from the Trojan to the output layer. This is a strict condition due to the following reasons. First, the trigger signal T has to switch between two consecutive simulations. Otherwise, the critical path will not pass through the trigger signal. Next, every signal in the critical path has to switch. Figure 5 shows an example to illustrate the difficulty of creating a critical path from the trigger T to the output layer. Assume that the payload A' flips from 0 to 1 due to the activation of the trigger. In order to propagate the delay, the signal P has to flip from 0 to 1, which requires signal N to have value 0 in the beginning. When we consider all the signals in a path from A' to the output layer, more and more constraints need to be applied. Directed test generation, such as ATPG or SAT-based approach, can be used to find the optimal solution when the payload is known. However, as we do not know the exact place of the trigger and payload a priori, these approaches may not work. In this section, we propose a probabilistic approach to increase the likelihood of constructing such a critical path using Hamming-distance based reordering.

Algorithm 2 shows our reordering approach to statistically create a critical path and maximize sensitivity. The main idea is to find a test vector that differs from the current test vector mostly as its successor. We define the *distance* of two vectors as the summation of two parts. The first part is the Hamming distance of the *feature vector*, which represents the activation status of all rare signals. For example, assuming a test t is

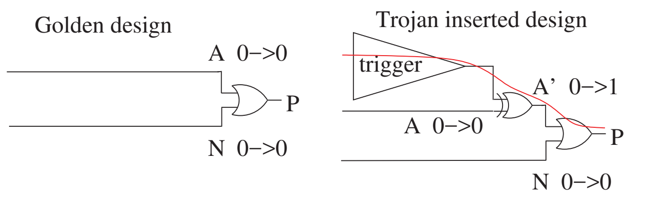

Fig. 5. There are many constraints to ensure a critical path from the trigger to the output layer. For example, signal N needs to be 0 in the first simulation.

able to activate the first three rare signals in Figure 1, then its feature vector is 1110. With a larger difference in the feature vector of two test vectors, one trigger condition is less likely to be activated by the two vectors simultaneously. The second part is the Hamming distance of the test vectors. Large Hamming distance between the test vectors increases the probability of signal switches in the cone area impacted by A'. As a large difference in the features vectors of two tests t_i and t_j typically implies a large Hamming distance of these two test vectors, we add a small weight (0.1 in Algorithm 2) to the Hamming distance of test vectors (the latter part). As shown in Algorithm 2, we first simulate the design and compute the feature vector for all test vectors. For each test vector t_i, we try to find the test vector with the largest distance among the remaining ones as its successor (line 7-16). After finding the test vector, we swap it with t_{i+1} (line 17).

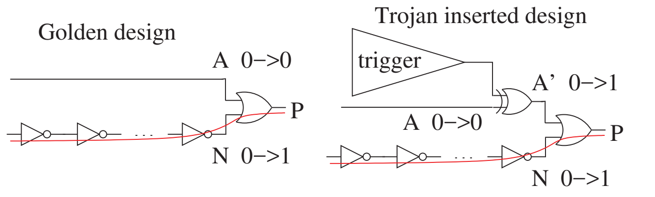

Fig. 6. The delay from the trigger may be masked by any signal in the path from A' to the output layer with longer delays.

The Hamming-distance based reordering is efficient, with k simulations and $O(k^2)$ computations of Hamming distance, where k is the number of generated test patterns. As the Trojan is unknown, the generated tests may not be able to sensitize the critical path from A' to the output layer. For example, when the signal N in the longer path switches, the delay of P is determined by N, which masks the delay from the Trojan as shown in Figure 6. As a result, there would be no difference between the delays from golden design and Trojan inserted design. In general, for some path from A' to the output layer, all neighbor signals with longer delays need to remain the same value. However, without knowing the exact Trojan, this requirement is hard to fulfill. Fortunately, as an attacker is likely to construct a hard-to-activate trigger condition, the path from the input layer to the trigger T is typically long. It potentially produces large delay in the trigger signal T, which leads to detection of Trojans as demonstrated in Section IV.

IV. Experimental Results

A. Experimental Setup

Implementation: All of our algorithms and simulators are implemented in C++. The SAT expressions in Algorithm 1 are solved using Z3 [35]. The experiments are conducted using a machine with Intel Xeon CPU E5-1620 v3 @ 3.50GHz and 16GB RAM.

Algorithm 2: Hamming-distance based Reordering

```
Input  : circuit netlist, test vectors $T = \{t_1, t_2, ..., t_k\}$
Output: reordered test vectors $T$
1  for $t_i$ in $T$ do
2      Simulate the netlist with $t_i$;
3      Set feature vector of $t_i$: each bit of $fv_i$ represents
         whether a certain rare signal is activated or not;
4  end
5  Set weight $\omega = 0.1$;
6  for $i = 1$ to $k$ do
7      Initialize best successor for $t_i$ as
         $bestSuccessor = -1$;
8      Initialize the largest distance as $maxdist = -1$;
9      for $j = i + 1$ to $k$ do
10         The distance of feature vector
             $dist1 = Hamming(fv_i, fv_j)$;
11         The distance of test vectors
             $dist2 = Hamming(t_i, t_j)$;
12         if $dist1 + \omega * dist2 > maxdist$ then
13             $maxdist = dist1 + \omega * dist2$;
14             $bestSuccessor = j$;
15         end
16     end
17     Swap the test vectors of $t_{i+1}$ and $t_{bestSuccessor}$;
18 end
19 Return $T$;
```

Benchmarks: To evaluate the effectiveness of our approach in detecting hardware Trojans, we selected five sequential benchmarks from ISCAS-89 [36], as well as a large benchmark MIPS from OpenCores [37]. Trojans are inserted in the same way as Figure 1, using rare signals to construct trigger conditions. For the two small ISCAS-89 benchmarks, s1196 and s1423, each trigger condition is constructed by 4 trigger points, while Trojans of the other benchmarks are constructed by 8 trigger points. All trigger points are selected from rare nodes from the design, where the rareness thresholds are 0.1 for ISCAS benchmarks and 0.005 for MIPS. The total number of rare nodes is listed in Table II. For each benchmarks, 1000 Trojans are randomly sampled. Each Trojan is inserted into the golden design to form one DUT. In other words, there are 1000 DUTs for each benchmark to evaluate the performance.

B. Path Delay Computation

The path delay can be measured using static timing analysis of gate-level models. We first compiled the benchmarks using Synopsys Design Compiler. Next, we generated Standard Delay Format (SDF) file that contains delay information of each gate and net in the design by linking with saed 90nm library [38]. Finally, SDF files are back-annotated into our simulator. The simulator simulates all DUTs with generated test patterns, and reports delay information computed using corresponding SDF files. Due to many factors in manufacturing steps, there are process variations in ICs, resulting in different delay fingerprints of the same design. To reflect the process variations, we added $\pm 7.5\%$ random variations to the SDF file of each DUT [25].

C. Evaluation Criteria

To evaluate the effectiveness of the generated tests by all approaches in detecting Trojans, we first simulated the golden design with the tests, and got the delay information of all registers. We use $dl_{gold}^{f}(t)$ to denote the delay for the register f of the golden design when simulating test pattern t. Then, we simulated each DUT with these tests, and got the delay information of all registers. Similarly, we use $dl_{dut}^{f}(t)$ to denote the delay for the register f in the DUT when simulating test pattern t. Finally, the maximum difference between the two delays which belong to the same register f is reported as our metrics to evaluate the performance of the tests from all approaches in (1).

$$\mathit{diff} = \max_{t,f}(|dl_{dut}^{f}(t) - dl_{gold}^{f}(t)|) \tag{1}$$

Assume that the test vector t^* produces the maximum delay difference in the register f^* for a given DUT, i.e., achieves the largest metric in (1). We define the following symbols for the ease of illustration:

- **OrigDelay:** the delay of f^* in the goden design when applying t^*, i.e., $dl_{gold}^{f^*}(t^*)$.
- **Sensitivity:** the relative difference of delays in golden design and DUT, i.e., $\mathit{diff}/dl_{gold}^{f^*}(t^*)$

D. Statistical Evaluation

Table I summarizes experimental results from the application of our approach on the benchmarks compared to random test vectors and ATPG test vectors. For random simulation, we generated 10K random vectors for each benchmark. The number of random test vectors is selected to balance the overall performance and simulation time. To generate ATPG test vectors for path delays, we utilized TetraMAX with all delay faults and full sequential mode. For our approach, we fix the number of test vectors to be 1000 for all benchmarks, i.e., $k = 1000$ for Algorithm 1. For each approach, Table I summarizes the number of test vectors (#), OrigDelay, delay difference (diff), and the average sensitivity over 1000 randomly sampled Trojans. From the results, we can see that random test vectors and ATPG achieve high delay sensitivity in small designs. However, the sensitivity produced by these two approaches are within 5% for two large benchmarks s38417 and MIPS, which is typically introduced by the noise. In contrast, our approach is able to achieve high sensitivity consistently. Overall, our approach can achieve 16 and 18 times improvement of sensitivity in delay based side-channel analysis over random test vectors and ATPG, respectively.

With the huge improvements in delay difference, our approach is able to detect more Trojans. In this experiment, a simple approach is used to declare the existence of a Trojan: if the delay in a DUT deviates from the delay in the golden design by more than the noise threshold (7.5%), then we declare that a Trojan exists in the DUT. Figure 7 shows the number of detected Trojans by these approaches. Among the 1000 randomly sampled Trojans, random simulation and ATPG are able to detect reasonable number of Trojans in the small design. However, the performance of these two approaches is poor for large designs, which detect less than 3% of all Trojans. On the other hand, our approach is able to detect more than half of the all Trojans for all benchmarks.

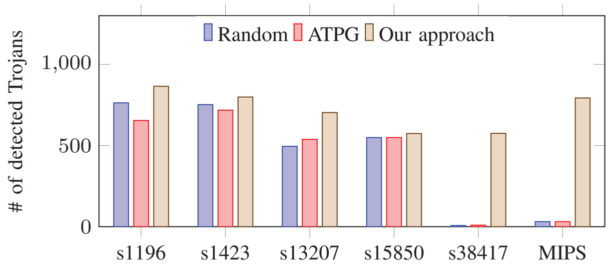

Fig. 7. The number of detected Trojan given the noise of $\pm 7.5\%$ noise.

Note that the performance of random simulation and ATPG is becoming worse when the design becomes large. It is due to the fact that the path between the input layer and the output layer in the small designs are relatively small, typically consisting of less than 10 gates. Therefore, an extra XOR gate from the Trojan can introduce reasonable delay difference to the delay of the output layer compared to 7.5% noise. However, with the number of gates increases in the paths, the effect of an extra gate becomes negligible. In contrast, our approach achieves consistent good performance in the all designs, due to the selection of test vectors that are likely to change the critical paths for the output layer entirely, as shown in Figure 4. In large designs, the change of critical paths is more likely to introduce drastically different delays.

The running time of our approach is shown in Table II. The results show that our approach is efficient in generating test vectors for both ISCAS benchmarks and MIPS. For all benchmarks except for s38417, the total test generation time is within 20 minutes. This relatively longer time for s38417, which is less than 2 hours, is because that the number of rare nodes in s38417 is more than three times the number of rare nodes in all the other benchmarks. Overall, our approach can generate 1000 test vectors efficiently.

One major problem of gate-level simulation is the slow simulation speed. Therefore, the compactness of generated tests is critical to reduce the overall debug time. When the generated test patterns are not compact, it usually consumes a lot more time in simulation than in generating tests. From Table I, 1000 test vectors generated by our approach are significantly better than 10K random vectors in both coverage and compactness of tests. While the tests generated by ATPG are slightly more compact in small benchmarks, its performance is the worst among the three approaches.

V. Conclusion

Hardware Trojans are threats to assets in integrated circuits. To detect hardware Trojans, side-channel analysis is a widely

TABLE I
PERFORMANCE COMPARISON OF OUR APPROACH WITH RANDOM SIMULATION AND ATPG OVER 1000 RANDOMLY SAMPLED TROJANS.

	Random				ATPG				Our Approach					
benchmark	#	OrigDelay (ps)	*diff* (ps)	sensi-tivity	#	OrigDelay (ps)	*diff* (ps)	sensi-tivity	#	OrigDelay (ps)	*diff* (ps)	sensi-tivity	impro. /Random	impro. /ATPG
s1196	10K	1347	702	52%	221	1622	415	26%	1000	1073	1221	114%	2.2x	4.4x
s1423	10K	1586	313	20%	103	1385	173	12%	1000	675	1456	216%	11x	17x
s13207	10K	2108	169	8%	411	1553	144	9.3%	1000	1478	931	63%	7.9x	6.8x
s15850	10K	2370	192	8.1%	472	2149	178	8.3%	1000	2249	682	30%	3.7x	3.7x
s38417	10K	31826	1279	4%	1169	28729	1161	4%	1000	14768	11738	80%	20x	20x
MIPS	10K	62998	2495	4%	1363	61751	2446	4%	1000	21156	18227	86%	22x	22x
Average	**10K**	**17039**	**858**	**5%**	**623**	**16198**	**753**	**4.6%**	**1000**	**6900**	**5709**	**83%**	**16x**	**18x**

TABLE II
TEST GENERATION TIME OF OUR APPROACH IN ALL BENCHMARKS.

bench	#gates	#wires	#rare	Algo. 1	Algo. 2	total
s1196	550	568	195	33.6s	0.2s	33.8s
s1423	456	502	50	26.5s	0.05s	26.6s
s13207	2335	2504	604	150.8s	0.5s	151s
s15850	2812	3004	649	352s	0.5s	353s
s38417	23815	23844	3103	6195s	2.4s	6197s
MIPS	18123	18343	906	1058s	1s	1059s
Average	**8015**	**8128**	**918**	**1303s**	**0.8s**	**1304s**

used approach. Existing path delay based side-channel analysis techniques are not effective since the difference in path delays between the golden design and Trojan-inserted design is negligible compared to process variation and environmental noise margins. We presented an automated test generation approach to take advantage of logic testing in maximizing the difference in path delays. Compared to existing research efforts that fixes one critical path, our approach explores two different critical paths for the same register in the two designs, resulting in significantly large difference in path delay. Our experimental results using a diverse set of benchmarks demonstrated that our approach outperforms state-of-the-art path delay based side-channel analysis techniques. Specifically, our approach is able to detect most of the Trojans while state-of-the-art techniques fail to detect most of them in large designs when process variation and noise margin is higher than 7.5%.

REFERENCES

[1] M. Tehranipoor and F. Koushanfar, “A survey of hardware trojan taxonomy and detection,” in *IEEE Design Test of Computers*, 2010.

[2] P. Mishra, S. Bhunia, and M. Tehranipoor, *Hardware IP Security and Trust*, 1st ed. Springer Publishing Company, Incorporated, 2017.

[3] R. S. Chakraborty et al., “MERO: A Statistical Approach for Hardware Trojan Detection” in *CHES*, 2009.

[4] A. Waksman et al., “Fanci: Identification of stealthy malicious logic using boolean functional analysis,” in *SIGSAC*, 2013.

[5] H. Salmani et al., “A novel technique for improving hardware trojan detection and reducing trojan activation time,” *TVLSI*, 2012.

[6] T. F. Wu et al., “Tpad: Hardware trojan prevention and detection for trusted integrated circuits,” *TCAD*, 2016.

[7] Y. Lyu and P. Mishra, “Automated Trigger Activation by Repeated Maximal Clique Sampling”, in *ASPDAC*, 2020.

[8] A. Ahmed, F. Farahmandi and P. Mishra, “Directed Test Generation using Concolic Testing of RTL Models,” in *DATE*, 2018.

[9] A. Ahmed and P. Mishra, “QUEBS: Qualifying Event Based Search in Concolic Testing for Validation of RTL Models,” in *IEEE International Conference on Computer Design*, 2017.

[10] X. Qin and P. Mishra, “Scalable Test Generation by Interleaving Concrete and Symbolic Execution,” in *VLSI Design*, 2014.

[11] Y. Lyu, X. Qin, M. Chen, and P. Mishra, “Directed Test Generation for Validation of Cache Coherence Protocols”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[12] F. Farahmandi and P. Mishra, “Automated Test Generation for Debugging Multiple Bugs in Arithmetic Circuits,” *IEEE Transactions on Computers (TC)*, vol. 68, no. 2, pp. 182-197, 2019.

[13] F. Farahmandi *et al.*, “Trojan localization using symbolic algebra,” in *Asia and South Pacific Design Automation Conference*, 2017.

[14] F. Farahmandi and P. Mishra, “Automated Test Generation for Debugging Arithmetic Circuits,” in *DATE*, 2016.

[15] A. Ahmed et al., “Scalable hardware trojan activation by interleaving concrete simulation and symbolic execution,” in *ITC*, 2018.

[16] Y. Lyu, A. Ahmed, and P. Mishra, “Automated Activation of Multiple Targets in RTL Models using Concolic Testing”, in *Design Automation and Test in Europe (DATE)*, 2019.

[17] Y. Lyu and P. Mishra, “Automated Test Generation for Activation of Assertions in RTL Models”, in *Asia and South Pacific Design Automation Conference (ASPDAC)*, 2020.

[18] J. Cruz et al., “An Automated Configurable Trojan Insertion Framework for Dynamic Trust Benchmarks”, in *DATE*, 2018.

[19] J. Cruz et al., “Hardware Trojan Detection using ATPG and Model Checking”, in *International Conference on VLSI Design*, 2018.

[20] X. Qin and P. Mishra, “Directed Test Generation for Validation of Multicore Architectures,” *ACM Transactions on Design Automation of Electronic Systems*, volume 17, no 3, article 24, 21 pages, 2012.

[21] M. Chen, P. Mishra and D. Kalita, “Automatic RTL Test Generation from SystemC TLM Specifications”, *ACM TECS*, 2012.

[22] M. Chen and P. Mishra, “Property Learning Techniques for Efficient Generation of Directed Tests,” *IEEE Transactions on Computers (TC)*, 60(6), pages 852-864, 2011.

[23] M. Chen and P. Mishra, “Functional Test Generation using Efficient Property Clustering and Learning Techniques,” *IEEE Transactions on CAD (TCAD)*, 29(3), 396-404, 2010.

[24] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, “Trojan detection using ic fingerprinting,” in *SP*, 2007.

[25] Y. Jin and Y. Makris, “Hardware trojan detection using path delay fingerprint,” in *HOST*, 2008.

[26] J. Aarestad et al., “Detecting trojans through leakage current analysis using multiple supply padI_{ddq}s,” *TIFS*, 2010.

[27] B. Cha and S. K. Gupta, “Trojan detection via delay measurements: A new approach to select paths and vectors to maximize effectiveness and minimize cost,” in *DATE*, 2013.

[28] X. Ngo et al., “Hardware trojan detection by delay and electromagnetic measurements,” in *DATE*, 2015.

[29] D. Ismari et al., “On detecting delay anomalies introduced by hardware trojans,” in *ICCAD*, 2016.

[30] Y. Huang et al., “Scalable test generation for trojan detection using side channel analysis,” *TIFS*, vol. 13, no. 11, pp. 2746-2760, 2018.

[31] Y. Huang et al., “MERS: Statistical test generation for side-channel analysis based trojan detection,” in *CCS*, 2016.

[32] Y. Lyu and P. Mishra, “A Survey of Side-Channel Attacks on Caches and Countermeasures”, *Journal of Hardware and Systems Security*, 2018.

[33] Y. Lyu and P. Mishra, “Efficient Test Generation for Trojan Detection using Side Channel Analysis”, in *DATE*, 2019.

[34] J. Li and J. Lach, “Negative-skewed shadow registers for at-speed delay variation characterization,” in *ICCD*, 2007.

[35] L. Moura and N. Bjørner, “Z3: An efficient smt solver,” *TACAS* 2008.

[36] “ISCAS89 sequential benchmark circuits,” https://filebox.ece.vt.edu/~mhsiao/iscas89.html.

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

[38] “Saededk90core - 90nm digital standard cell li-brary,” http://www.synopsys.com/community/universityprogram/pages/library.aspx.