Test Generation using Reinforcement Learning for Delay-based Side-Channel Analysis

Zhixin Pan, Jennifer Sheldon and Prabhat Mishra
Department of Computer & Information Science & Engineering
University of Florida, Gainesville, Florida, USA

ABSTRACT

Reliability and trustworthiness are dominant factors in designing System-on-Chips (SoCs) for a variety of applications. Malicious implants, such as hardware Trojans, can lead to undesired information leakage or system malfunction. To ensure trustworthy computing, it is critical to develop efficient Trojan detection techniques. While existing delay-based side-channel analysis is promising, it is not effective due to two fundamental limitations: (i) The difference in path delay between the golden design and Trojan inserted design is negligible compared with environmental noise and process variations. (ii) Existing approaches rely on manually crafted rules for test generation, and require a large number of simulations, making it impractical for industrial designs. In this paper, we propose a novel test generation method using reinforcement learning for delay-based Trojan detection. This paper makes three important contributions. 1) Unlike existing methods that rely on the delay difference of a few gates, our proposed approach utilizes critical path analysis to generate test vectors that can maximize the side-channel sensitivity. 2) To the best of our knowledge, our approach is the first attempt in applying reinforcement learning for efficient test generation to detect Trojans using delay-based analysis. 3) Our experimental results demonstrate that our method can significantly improve both side-channel sensitivity (59% on average) and test generation time (17x on average) compared to state-of-the-art test generation techniques.

CCS CONCEPTS

• **Security and privacy → Side-channel analysis**; • **Computing methodologies → Machine learning algorithms**.

KEYWORDS

Test Generation, Side-Channel Analysis, Reinforcement Learning

ACM Reference Format:
Zhixin Pan, Jennifer Sheldon and Prabhat Mishra. 2020. Test Generation using Reinforcement Learning for Delay-based Side-Channel Analysis. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '20), November 2–5, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3400302.3415710

1 INTRODUCTION

With the rapid development of semiconductor technologies coupled with increasing demands of complex *System-on-Chips* (SoCs), the vast majority of semiconductor companies utilize global supply chains. A long and distributed supply chain provides opportunity for third-party Intellectual Property (IP) vendors as well as service providers to implant Hardware Trojans (HT) inside SoCs [8, 21, 23, 24]. Therefore, Trojan detection is widely acknowledged as a major focus to enable secure and trustworthy SoCs.

Existing Trojan detection techniques can be broadly classified into two categories: logic testing and side-channel analysis. Logic testing methods such as Automatic Test Pattern Generation (ATPG) [2] or statistical test generation [4, 15] try to activate Trojans using generated tests, but they have two major limitations: (1) They suffer from high computational complexity for large designs. (2) Since it is infeasible to generate all possible input patterns, the generated tests are not effective in activating stealthy Trojans. The triggering conditions for Trojans are usually crafted as a combination of rare conditions, such that Trojan-implanted designs will retain exactly the same functionality as golden designs until a rare condition is satisfied to yield malicious behavior. Figure 1 shows a trivial example from ISCAS'85 benchmark. In this design, both F and G are signals with '0' as the rare value. The shaded AND gate with inverted inputs will only be triggered if both F and G become '0', and, once triggered, the succeeding XOR gate will invert signal I, which is called an *asset.*

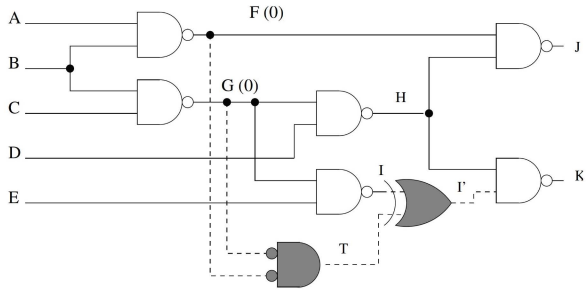

Figure 1: Example of HT triggered by rare signals

Side-channel analysis is a promising alternative since it compares the difference of side-channel signatures (such as path delay, electromagnetic emanation, dynamic current, etc.) with the expected (golden) values to detect Trojans. However, the effectiveness of side-channel analysis depends on the HT's side-channel leakage. The noise induced by the environment or process variation usually overshadows the Trojan footprint, which makes the detected difference negligible. Recent efforts have tried to combine logic testing and side-channel analysis [11, 12, 16, 18, 19] in order to improve side-channel sensitivity. Specifically, their goal is to maximize activity in suspicious regions while minimizing the activity in the rest of the design. While existing approaches provide promising avenues, they face two major challenges. First, the test generation time complexity is exponential for these methods, which severely limits their usability. Second, the side-channel difference achieved by these approaches is not large enough to offer high confidence in HT detection results.

In this paper, we address these challenges by proposing a novel test generation approach using reinforcement learning for delay-based side-channel analysis. The remarkable success of machine

ICCAD '20, November 2–5, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8026-3/20/11…$15.00
https://doi.org/10.1145/3400302.3415710

learning (ML) in a variety of hardware security tasks [5, 6, 10] has inspired us to explore its potential in HT detection. This paper makes three important contributions.

(1) We develop an automated and efficient test generation method using reinforcement learning to maximize the difference in path delay between the Trojan-implanted design and golden design by exploiting critical path analysis.
(2) To the best of our knowledge, our approach is the first attempt in applying reinforcement learning for test generation to detect Trojans using delay-based side-channel analysis.
(3) Our experimental evaluation applied on several benchmarks demonstrates that our method outperforms the state-of-the-art technique in both test generation time efficiency (17x on average) and side-channel sensitivity (59% on average).

The rest of this paper is organized as follows. We survey related efforts in Section 2. Section 3 describes our proposed method on efficient test generation using reinforcement learning. Section 4 presents experimental results. Finally, Section 5 concludes the paper.

2 RELATED WORK

2.1 Delay-based Side-Channel Analysis

There are various physical signatures of electronic devices suitable for side-channel analysis, such as path delay [20], dynamic current [19] and electromagnetic emanations [14]. Among them, we target path delay for three major reasons.

(1) *Independence:* The delay between any gates in the design can be measured independently, which provides more comprehensive information compared to other side-channel signals.
(2) *Diversity:* Implanted Trojans can impact path delay in multiple ways. Let us consider the design in Figure 1 as an example. There will be an increase of propagation delay for the gates producing signals F and G, since they are connected to an extra gate, which leads to increased capacitive load. Second, since one XOR gate and one AND gate were inserted to deliver the payload, the path delay will always have at least two gates difference from the golden design for any paths through these inserted XOR/AND gates.
(3) *Stability:* Delay-based Trojan detection techniques provide superior performance under parameter variations by leveraging statistical techniques [22]. This stability guarantees high confidence of detection results from delay-based analysis.

However, crafting test vectors that can reveal the impact of implanted Trojans on path delay is not a trivial task. Traditional approaches utilize static analysis where they enumerate all possible paths (removing unrelated paths) and exploit ATPG [7] to find feasible input patterns to trigger the desired path. There are two fundamental limitations of these approaches:

(1) The computational complexity grows exponentially with the design size. It is time consuming and even impractical for large designs due to exponential nature of possible paths.
(2) The detection result is extremely sensitive to environmental noise since the differences in delay induced by these methods are often negligible. Without activation of the HT, the difference is usually from a few gates (e.g., only two gates in Figure 1). In fact, even with Trojan successfully triggered, they are not guaranteed to generate a critical path from the Trojan to an observable output for propagating the delay.

Our proposed approach addresses these challenges by utilizing critical path analysis. It significantly increases the path delay difference induced by implanted HTs, and is able to achieve better results, which is discussed in Section 3.

2.2 Reinforcement Learning

Reinforcement Learning (RL) has shown its potential in solving complex optimization problems [13, 17, 26]. Searching for optimal test vectors in target designs to maximize the side-channel sensitivity can be viewed as an optimization problem. RL is a branch of machine learning, but unlike the commonly-known supervised learning, it is closer to human learning. For example, although an infant baby is not able to understand spoken words, it can still master language after a period of exploration. This exploration process is actually a process of gradually learning the rules (lexicon and grammar) of speaking through trials and responding to feedback from the environment. Similarly, RL also learns to find an optimal strategy through a series of attempts and constantly adjusts its behavior based on the feedback.

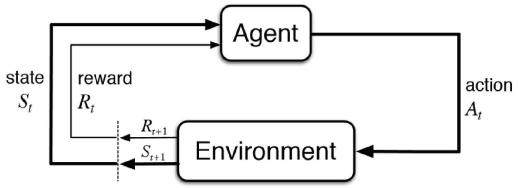

Figure 2: The basic framework of reinforcement learning. At time stamp *t*, state S_t and reward R_t are fed into Agent, which produce action A_t with updated strategy. The agent interacts with the environment to obtain new state S_{t+1} and reward R_{t+1}, then starts the next round of learning process.

An overview of RL framework is shown in Figure 2. It consists of five core components: *Agent*, *Environment*, *Action*, *State* and *Reward*.

- ***Agent*** refers to the object that can interact with the environment through actions. The agent of reinforcement learning is usually the set of test cases to be optimized, which is continuously updated through learning process.
- ***Environment*** is the receiver of the action, such as the optimization problem itself.
- ***Action*** consists of all possible operations that may affect the environment, such as using the current strategy for one-step calculation.
- ***State*** refers to information about the environment that can be perceived by the Agent, such as conditions and parameters.
- ***Reward*** is the feedback information from the environment that describes the effect of the latest action. For optimization problems, it often refers to the gain of objective function after performing the current operation.

The process of reinforcement learning is a process of obtaining feedback by interacting with the environment, and then adjusting the actions based on the feedback in order to maximize the total reward. The goal of reinforcement learning is to find an optimal strategy to maximize the rewards obtained during the entire interaction process. In terms of implementation, reinforcement learning is a process of gradually optimizing the parameters of the algorithm through multiple rounds to enhance the learning effect. To the best of our knowledge, our work is the first attempt in utilizing reinforcement learning for side-channel-analysis-aware test generation.

3 PROPOSED METHOD

3.1 Motivation

The difficulty of delay-based side-channel HT detection comes from designing proper test vectors to increase the observability of side-channel differences. Specifically, the test vectors should be able to reveal the impact of an inserted Trojan on path delay as much as possible. Existing approaches have focused on passively enumerating possible paths affected by the HT. But if the HT is not triggered, only a few gates difference can be obtained, which is hard to distinguish from environmental noise. Consequently, the detection results are not promising. In contrast, our proposed solution focuses on exploring the impacts through critical path analysis, which allows us to actively change the critical path and magnify side-channel differences.

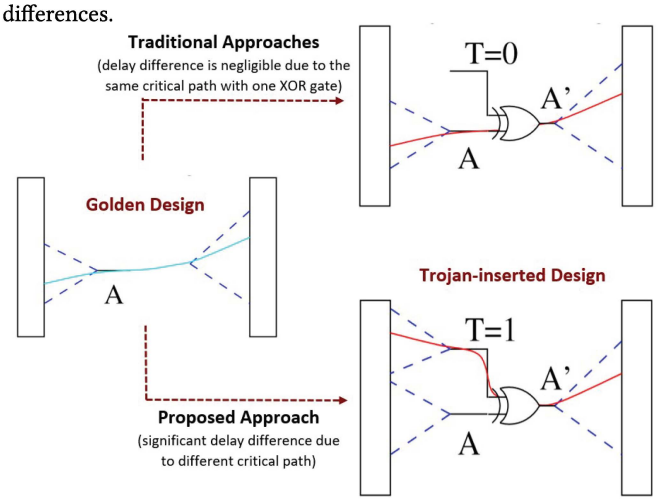

Figure 3: Maximizing delay difference by changing critical path [20]. By triggering T, the critical path from input layer to A' is significantly changed.

Figure 3 shows that the activation of a trigger T is necessary for maximizing the delay difference. The top part of the figure shows that if the test vector fails to activate the Trojan ($T = 0$), the critical path from the input layer towards the Trojan is exactly the same as that in the golden design. Then the delay difference is limited to the inserted gates themselves (e.g., only one XOR gate in the figure). The bottom part of the figure indicates that the critical path will be drastically different if the trigger can be activated. Note that the trigger signal T has to switch between consecutive input patterns, otherwise there will be no contribution from the Trojan to the path delay because the related signals remain the same between two consecutive tests. Once the above requirements are satisfied, a completely different critical path from input layer to Trojan is obtained, so that we can expect a huge difference between the measured delay differences. Consequently, the big problem of test generation now is divided into two sub-problems: how to find a good initial test for triggering the Trojan, and how to efficiently generate proper succeeding tests to switch triggering signals. Due to their stealthy nature, HTs are very likely to be activated by rare triggered conditions, therefore, the two sub-problems can be transformed into:

- Generate initial tests for triggering rare nodes
- Generate succeeding tests for triggering rare switches

There are major research challenges in solving the above two sub-problems. If we shift focus to a succeeding path (a path from the Trojan to the output layer in which the delay is propagated through the design) starting from node A in Figure 3, there has to be a critical path from A' to output layer to propagate the delay. Otherwise the delay is cut off and hidden from all other nodes succeeding A'. Unfortunately, creating such path is an NP-hard problem because, in the worst-case scenario, every signal in the critical path has to be taken into consideration. For example, in Figure 4, the trigger signal T switches from 0 to 1. Signal C, whose original value is 0, can either switch to 1 for propagating the delay induced by the switch through the AND gate to E or remain unchanged to propagate it through the XOR gate to D. This process will recursively continue for D and E to calculate constraints if they are chosen to be added to the critical path. The search space grows exponentially, bringing in numerous constraints. Therefore, traditional SAT-based approach is not feasible here, and we plan to apply reinforcement learning to tackle this issue. We will discuss this application in detail in Section 3.4.

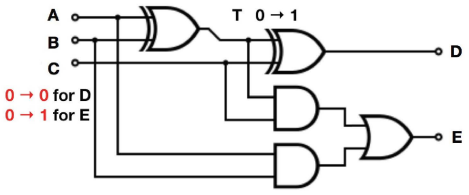

Figure 4: When the trigger signal T switches, C has to make a decision to propagate this change through either D or E. This process continues since D and E have to consider how to propagate their value changes to the output layer.

3.2 Overview

Figure 5 shows an overview of our proposed method. The primary goal is to generate a sequence of test patterns $(t_1, t_2, ..., t_n)$ such that for every consecutive pair of tests (t_i, t_{i+1}), the delay-based side channel sensitivity is maximized. For a given circuit design, we first obtain a set of proper initial test cases to ensure triggering rare nodes (Section 3.3). Next, those initial test cases in previous step are fed into an reinforcement machine learning model as initial inputs, which is trained with a stochastic learning scheme(Section 3.4) to increase the probability of triggering rare switches. After sufficient iteration of training, a well-trained RL model is exploited for automatic test generation. It starts working with initial input patterns, and utilizes the newly generated test vectors as input in the next round to continuously generate a sequence of test patterns of the desired amount.

3.3 Generation of Initial Vectors

As discussed in Section 3.1, it is important for test patterns to activate trigger conditions because if the test pattern fails to activate the Trojan, the delay of the golden design and the Trojan-inserted design differs by, at most, one gate. Therefore, our goal is to maximize the probability of activating trigger conditions. Since an attacker is likely to construct trigger conditions based on rare nodes to avoid detection, we need to generate initial vectors that can maximize the activation of rare nodes. Algorithm 1 shows the major steps in generating efficient initial vectors. It accepts the design (circuit netlist), the list of rare nodes (R) as well as the number of test vectors (n) as inputs, and produces n test vectors. The number of vectors presents a trade-off – a larger number can lead to longer runtime, but it is likely to improve the probability of activating trigger conditions compared to a smaller number of initial vectors.

Algorithm 1 first computes the logic equations for each rare node by analyzing the cone-of-influence for each of them. Next, it generates n test vectors, one in each iteration. The test generated in

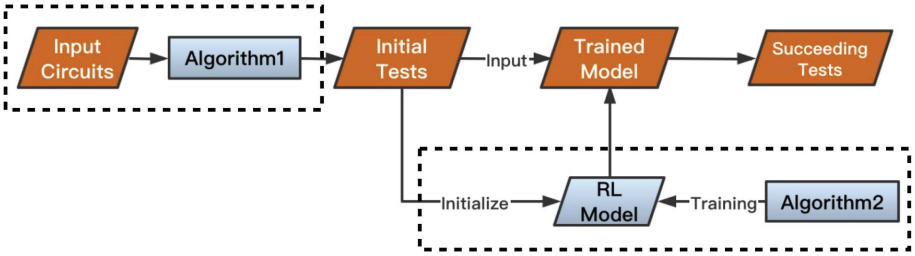

Figure 5: Our proposed framework consists of two major activities: initial test generation and reinforcement learning.

each iteration is expected to activate different rare nodes since we randomize the order of the rare nodes. Algorithm 1 also tries to a generate a test vector that can cover a large number of rare nodes. We ensure this by adding as many rare nodes as possible without making an invalid trigger. We use a SAT solver [9] to find a test that would activate all the nodes in the trigger simultaneously. Since an attacker is likely to create a trigger with the smallest possible number of rare nodes to minimize the path delay footprint, tests generated by our approach have a higher likelihood of detecting small triggers. In other words, if we can find a test that can cover a trigger with a large number of nodes, although we do not know the actual trigger inserted by an attacker, the likelihood of activating that Trojan goes up if we assume that the actual trigger will consist of a small number of nodes because the small trigger signals are a subset of the large trigger signals. These test cases will be used by the reinforcement learning (RL) model to form the initial state. The RL model is able to automatically search for the best succeeding test patterns, as discussed in the next section.

Algorithm 1: Generation of Initial Test Patterns

Input : Design (D), Rare nodes (R), Number of initial vectors (n)
Output: Test Patterns

Compute logic equations for each rare node in D
Initialize $Tests$={}
$i = 1$
repeat
  Trigger $TR = \emptyset$
  Randomize the order of rare nodes R
  for *each rare node* $r \in R$ **do**
    if $TR \cup r$ *is a valid trigger* **then**
      $TR = TR \cup r$
  Solve TR and get a $test_{TR}$
  $Tests = Tests \cup test_{TR}$
  $i = i + 1$
until $i > n$;
Return $Tests$

3.4 Generation of Succeeding Vectors

There are two crucial requirements for a succeeding test vector (a test vector produced by the RL model, which can then be used to generate another succeeding, or consecutive, test vector). First, the rare signals triggering Trojans have to switch between consecutive test patterns. If there is no rare switch, the critical path will not pass through the trigger signal. Second, the optimal succeeding test vector should be able to produce a critical path from the Trojan to the output layer which is completely different from the path in the golden design. Otherwise, the delay difference created by a Trojan cannot be propagated, and the maximum delay difference is suppressed.

For the first requirement, a SAT-based algorithm can solve for possible vectors to satisfy the rare switches. But the second condition, creating a critical path from the Trojan to the output, is an NP-hard problem, as discussed before. Traditional approaches have failed to satisfy this demand because exploiting ATPG or SAT is expensive for large circuits. Moreover, strict conditions are required for these approaches to function. One such condition is a rough estimate on the actual Trojan payload. Even in the state-of-the-art method, ATGD [20], the author circumvents this task by choosing to perform a test reordering. The author generates a large number of test patterns, and then performs a Hamming-distance-based reordering of these patterns with the expectation that the large Hamming distance increases the probability of signal switches in the cone area. This approach introduces significant time complexity in both steps. The first step is time-intensive since it needs to consider a large number of initial vectors to produce reasonable results. The reordering step requires quadratic time complexity in terms of the number of initial vectors. The author makes several heuristic assumptions to increase the probability of constructing a critical path between the Trojan and output layer; some of these assumptions may not be valid in many scenarios.

In order to address this fundamental challenge, we plan to apply reinforcement learning (RL) to enable automatic succeeding test generation. We explored the effectiveness of Hamming-distance-based analysis for satisfying the requirements of succeeding test vectors, and this analysis will be deployed as a component of the loss function in our model. A prototype version of our basic workflow is illustrated in Figure 6. This workflow is very similar to the scheme illustrated in Figure 2.

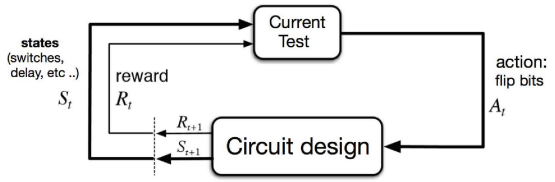

Figure 6: The reinforcement learning model for automatic test generation. At time stamp *t*, the model takes action A_t on current test case to flip its bits. Then the mutated test will interact with the circuit design (environment), while feedback S_{t+1} and reward R_{t+1} are sent back to evaluate the new test case. The RL model gradually learns the optimal strategy for flipping bits of previous test vectors, so that the newly generated test pattern can maximize the reward received.

Here, the RL components are matched as:

- ***Agent***: The current test vector.
- ***Environment***: The circuit design.
- ***Action***: Mutation of current test vector consisting of a sequence of bit flipping operations on each bit.

- ***State***: Activation of trigger nodes and rare switches.
- ***Reward***: Evaluation that produced a succeeding test vector.

Training of proposed model faces three serious challenges:

(1) The possible number of actions is exponential. For a test vector of length n, there are $2^n - 1$ possible ways to flip its bits to create a different test case.
(2) It is hard to determine the exact reward value of each operation. Because actions like "flipping the second bit" can either increase the difference of path delay or do exactly the opposite of another initial pattern.
(3) This naive learning framework cannot prevent an "infinite loop" from happening. That is, throughout learning, the model could consider v_2 as the best successor for v_1. It could also happen that v_1 maximizes the reward if it follows v_2. Then this $\{v_1, v_2, v_1, v_2, ...\}$ repetitive loop can continue forever. Finally, the test set produced is useless since it only consists of two individual test patterns.

We apply a stochastic approach to train our RL model to solve these challenges. In traditional value-based reinforcement learning processes, succeeding vectors are deterministic since the choice of action is fixed for a given state to maximize the reward. But in our method, a stochastic scheme is applied. First, for each step of learning, the action is chosen randomly, i.e, for each bit of current test vector, a probabilistic selection will determine whether to flip it or not. This non-deterministic result is not completely arbitrary but determined by a series of probability distributions. Second, the basic principle of our method is to adjust these probability distributions based on the expectation of reward. Specifically, when positive reward expectation is obtained, the probability of the corresponding action is increased, and vice versa.

This stochastic approach not only ensures non-determinism but also avoids blindness action reward scheduling, which is the key barrier to the general training approach. In addition to the theoretical advantages of dealing with optimization problems, this strategy also possesses huge advantages in implementation since, for a test vector of length n, there is no need to encode all $2^n - 1$ possible mutations. There is only a need to maintain a table consisting of n binomial distributions.

As a result, all above-mentioned challenges are addressed properly. There is no longer a time cost for redundant test generation in previous steps. Furthermore, the reward value is the expectation of action rather than a fixed value. Consequently, the infinite loop problem no longer exists since bit flipping is probabilistically determined.

The learning process is shown in Figure 7, which is basically a strategy optimization process. At the beginning, randomly initialized probability distributions are assigned to each bit of the test vector. Of course, there is no guarantee for this strategy to generate promising results, so the newly generated test case will likely to provide poor performance and receive negative rewards. The goal of learning is to improve the expected reward, which can be formulated in the following way:

$$J(\theta) = -E_{\tau \sim p_\theta}(R_\tau), \quad p_\theta(\tau) = p_\theta(\tau_1|\tau_2|...|\tau_n)$$
$$R_\tau = RS(t, t_\tau) + \lambda \cdot Hamming(t, t_\tau)$$
$$\theta^* = \text{argmin}\, J(\theta)$$

where action τ is the union of probabilistic flipping action for each bit, i.e $\{\tau_1, \tau_2, ..., \tau_n\}$. The reward R_τ of action τ is defined as a combination of rare switches $RS(t, t_\tau)$ and the Hamming distance $Hamming(t, t_\tau)$, an idea which we adopted from [20]. λ is a regularization factor. The loss function $J(\theta)$ is the expectation of reward since τ is chosen by

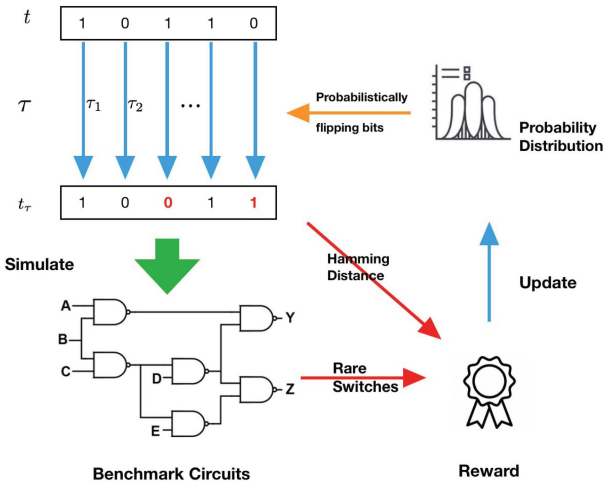

Figure 7: Illustration of proposed stochastic reinforcement learning method

probability distribution p_θ parametrized by θ. A negative sign is put ahead since we want to minimize this "loss" through gradient descent. Also, to circumvent the non-differentiability of this objective function, we resort to the standard REINFORCE learning rule[25] which gives an alternative gradient of $J(\theta)$ w.r.t. θ:

$$\nabla_\theta J(\theta) = -E_{\tau \sim p_\theta}[(\sum_{i=1}^{n} \nabla_\theta log\, p_\theta(\tau_i)) \sum_{j=1}^{n} R_{\tau_j}]$$

For each iteration, the model starts learning, and the product of the learning rate α and $\nabla_\theta J(\theta)$ is used to update the parameter until the expected reward exceeds a certain threshold or no longer increases: This training process is shown in Algorithm 2. The reinforcement learning framework enables generation of efficient tests for delay-based side-channel analysis, as demonstrated in the next section.

Algorithm 2: Stochastic training of RL Model

Input : Design(D), Model Parameter (θ), Initial tests (T), number of epochs k, learning rate α
Output: Optimal Model Parameter θ^*

Initialize probability distributions $P = P_\theta$
Initialize RL Model $M_\theta = init(T, P)$
$i = j = 0, n = size(T)$
repeat
  repeat
    for *each* $t \in T$ **do**
      τ = mutate(t, P)
      $R_\tau = RS(t, t_\tau) + \lambda \cdot Hamming(t, t_\tau)$
      $J(\theta) = -E_{\tau \sim p_\theta}(R_\tau)$
    Update parameter : $\theta = \theta + \alpha \nabla_\theta J(\theta)$
  until $j \geq n$;
until $i \geq k$;
Return θ

4 EXPERIMENTS

This section is organized as follows: First, we describe our experimental setup including implementation details as well as evaluation criteria. Next, we present our experimental results.

4.1 Experimental Setup

RL Implementation: The model training was conducted on a host machine with Intel i7 3.70GHz CPU, 32 GB RAM and RTX 2080 256-bit GPU. We developed Python (3.6.7) code using PyTorch (1.2.0) with cudatoolkit (10.0) as the machine learning library. The training process consisted of 200 epochs where we updated the learning rate α starting with 0.01, pushing it up to 0.2, and lowering it again to 0.02.

Hardware Implementation: For test simulation, we compiled each benchmark design (golden and Trojan-inserted) using Quartus Prime 18.0 Lite Edition in order to generate SDO (timing annotation similar to SDF) files associated with each benchmark design. Each SDO file was generated with the Cyclone IV-E FPGA to ensure that Verilog code constructs appearing in each benchmark were associated with the same hardware for timing. Next, we generated Verilog testbenches using the test vectors produced by our framework. The testbenches initialized the scan chains with suitable values from the test vectors, and then applied the primary inputs. We ran the tests sequentially with two clock cycles between test applications. We ran the testbenches using ModelSim version SE-64 2020.1's timing simulation capabilities with the Verilog benchmark and testbenches as well as the Quartus-generated SDO files. We recorded each simulation's data by generating an associated event list file in ModelSim.

Benchmarks: To demonstrate the test vectors' effect on different designs, we carried out the experiment on five benchmarks from ISCAS-89 [1].

Path Delay Computation: ModelSim event list files provide initial signal values and changes in signal values over the course of the simulation. To compute path delay, we subtracted the time between changes of the same signal for each application of a test.

Evaluation Criteria: To quantify the efficacy of test vectors, we collected the path delay data for simulated golden designs and designs with inserted Trojans. We then used this data to quantify the effect of the inserted Trojan on the path delay with the given test vector using the *difference*, here defined as:

$$difference = \max_{t,f}(|delay^{f}_{DUT}(t) - delay^{f}_{gold}(t)|) \quad (1)$$

where f is the set of all registers in the tested benchmark and t is the set of all tests in the analyzed test vector. The *difference* is the maximum path delay difference between golden and Trojan-inserted designs (designs under test or DUTs). We also adopt the "sensitivity" from [20] as a metric, which refers to the scaled delay difference between the DUT and golden design. The sensitivity is defined as:

$$sensitivity = difference/delay^{f^*}_{gold}(t*)) \quad (2)$$

where f^* refers to the register producing the maximum delay difference, and t^* refers to the test producing the maximum delay difference.

4.2 Evaluation Results

To demonstrate the quality of tests compared to existing approaches, we evaluate the following three different test generation schemes.

- *Random*: Random test generation, applied as the baseline.
- *ATGD*: State-of-the-art algorithm proposed in [20].
- *Proposed*: Our proposed method.

We generated 1000 random test vectors using all three approaches for each benchmark, and Table 1 summarizes the results of performance evaluation. We present the difference of delay and the average sensitivity for each configuration. From the results, we can see that our proposed method provides the best performance. For random test generation, there is a significant decrease in sensitivity with the increase of benchmark size. For example, when it comes to relatively large benchmark like s38417, the sensitivity is only around 4%, which can hide in typical environmental noise. The ATGD [20] is better than random simulation with an average sensitivity of 73.38%. Our proposed method provides superior results for all these benchmarks with an average sensitivity of 132.92%, which grants 60% extra sensitivity than ATGD. Also, ATGD cannot guarantee the stability of test quality. For s15850 (2812 gates), the sensitivity drops below 30%, while for s13207 with same level of scale (2335 gates), it achieved 72.24% sensitivity. This is expected since ATGD relies on a simple heuristics. In contrast, our proposed method consistently provides high sensitivity (e.g., 97.28% and 133.45% for these cases).

The benefit of improving sensitivity is directly reflected by the results of HT detection. We apply these delay number for HT detection by following the *threshold criteria*: when the delay difference between DUT and golden design exceed certain threshold, we claim the existence of HT inside the DUT. We use a 7% threshold in this paper based on the study [3] that provides an estimate on process variations and environmental noise margins. Figure 8 presents the rate(%) of HTs detected in each benchmark.

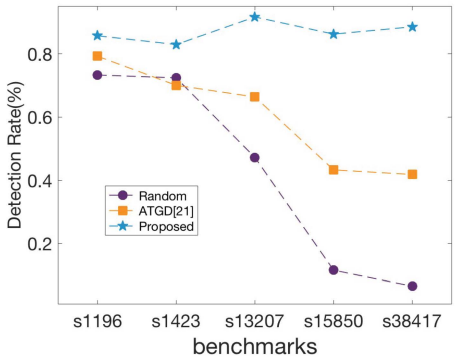

Figure 8: The performance of HT detection rate for all approaches on each benchmark.

As shown in Figure 8, when we consider tiny benchmarks, all approaches achieved a decent detection rate. Because the path between the input layer and the output layer in smaller designs is very short, even if these methods do not activate the Trojan, the extra inserted gates and change in capacitive load can still produce certain degrees of delay difference. However, when it comes to large benchmarks, the random test generation completely failed to detect most of the HTs. The ATGD performs better than random test generation, but it still faces the problem of decreasing detection rate with increasing design scale. In the worst case, over 50% of HTs successfully bypass detection by ATGD in s15850 and s38417, which is unacceptable. By comparison, the rate of detection by our proposed method is always above 80%. It also achieved a very high detection rate (88.54%) in the largest tested benchmark (s38417).

Another important factor of approach evaluation is the time complexity. Table 2 compares the running time between ATGD and our proposed approach deployed on each benchmark (Random approach is out of consideration since it is definitely the fastest one due to its no-calculation nature). The results show that our method can generate test vectors much faster than ATGD. The huge difference of time efficiency comes from the following reasons: In our experiment, the desired task is to generate 1000 test vectors for each benchmark. If case of ATGD, we need to exploit an SAT-based method to generate 1000 test vectors, then perform a reordering algorithm on these 1000

Table 1: Performance comparison with existing approaches

	Random			ATGD [20]			Proposed				
bench	golden delay(ps)	*difference* (ps)	sensitivity	golden delay(ps)	*difference* (ps)	sensitivity	golden delay(ps)	*difference* (ps)	sensitivity	impro. /Random	impro. /ATGD
s1196	1302	698	53.60%	982	1237	125.96%	1224	1590	129.99%	2.5x	1.1x
s1423	1625	275	19.23%	666	1368	205.40%	618	1840	297.73%	15.7x	1.46x
s13207	1911	143	7.48%	1621	996	70.09%	1254	1382	111.20%	14.9x	1.6x
s15850	2340	111	4.74%	2398	703	29.31%	2209	2149	97.28%	20.5x	3.4x
s38417	33319	1520	4.56%	12580	9088	72.24%	16579	22126	133.45%	20x	1.9x
Average	**8099**	**549**	**6.76%**	**3649**	**2678**	**73.38%**	**4377**	**5818**	**132.92%**	**15x**	**1.9x**

vectors to sort them. For an RL based approach, on the other hand, the SAT method is only applied to generate several vectors as candidates for initial states to be fed into the learning model. Meanwhile, the model training is composed of 200 iterations where each iteration is basically a one-step succeeding test generation and evaluation. When the model is well-trained, it can generate the remaining test vectors. So, as we can see, assuming k is the desired number of test vectors, our approach finishes the task with linear $O(k)$ time complexity, while, for ATGD, the reordering process requires a quadratic ($O(k^2)$) time complexity .

5 CONCLUSION

Hardware Trojans are a serious threat to designing trustworthy integrated circuits. While side-channel analysis is promising, existing delay-based techniques are not effective in detecting hardware Trojans. Specifically, existing approaches introduce high time complexity requiring extra computation resources, and are therefore not suitable for large designs. Most importantly, these approaches lead to small differences in path delay between the golden design and the Trojan-inserted design; this makes the approaches unreliable in the presence of environmental noise and process variations. In this paper, we proposed reinforcement-learning-based test generation for effective delay-based side-channel analysis. We generated a set of efficient initial patterns through SAT-based approach. We utilized reinforcement learning using stochastic methods to generate beneficial succeeding patterns. Our approach is fast, automatic, and significantly improves the side-channel sensitivity compared with existing research efforts. Specifically, our method takes, on average, 94% less time for generating 1000 test cases for each benchmark, and it is able to detect most implanted Trojans in all tested benchmarks. The state-of-the-art method, on the other hand, failed to detect 58% of Trojans on large designs.

Table 2: Comparison of Test Generation Time.

bench	#gates	#wires	ATGD [20]	Proposed	Speedup
s1196	550	568	35.2s	7.4s	4.75x
s1423	456	502	36.8s	7.3s	5x
s13207	2335	2504	203.7s	28.1s	7.3x
s15850	2812	3004	492s	66.9s	7.4s
s38417	23815	23844	6022.6s	282.6s	21.3x
Average	**8015**	**8128**	**1358s**	**79s**	**17x**

ACKNOWLEDGMENTS

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

REFERENCES

[1] [n. d.]. ISCAS89 Sequential Benchmark Circuits. https://filebox.ece.vt.edu/~mhsiao/iscas89.html.

[2] Alif Ahmed, Farimah Farahmandi, Yousef Iskander, and Prabhat Mishra. 2018. Scalable Hardware Trojan Activation by Interleaving Concrete Simulation and Symbolic Execution. In *IEEE International Test Conference, ITC 2018*. 1–10.

[3] Bharathan Balaji, , et al. 2012. Accurate Characterization of the Variability in Power Consumption in Modern Mobile Processors. In *Workshop on Power-Aware Computing and Systems*.

[4] Rajat Subhra Chakraborty et al. 2009. MERO: A Statistical Approach for Hardware Trojan Detection. In *Cryptographic Hardware and Embedded Systems*. 396–410.

[5] Mingsong Chen and Prabhat Mishra. 2010. Functional Test Generation Using Efficient Property Clustering and Learning Techniques. *IEEE Trans. on CAD of Integrated Circuits and Systems* 29, 3 (2010), 396–404.

[6] H. Choo et al. 2020. Register-Transfer-Level Features for Machine-Learning-Based Hardware Trojan Detection. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* 103-A, 2 (2020), 502–509.

[7] Jonathan Cruz, Farimah Farahmandi, Alif Ahmed, and Prabhat Mishra. 2018. Hardware Trojan Detection Using ATPG and Model Checking. In *International Conference on VLSI Design*. 91–96.

[8] Jonathan Cruz, Yuanwen Huang, Prabhat Mishra, and Swarup Bhunia. 2018. An automated configurable Trojan insertion framework for dynamic trust benchmarks. In *Design, Automation & Test in Europe Conference (DATE)*. 1598–1603.

[9] Leonardo Mendonça de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS*. 337–340.

[10] Rana Elnaggar and Krishnendu Chakrabarty. 2018. Machine Learning for Hardware Security: Opportunities and Risks. *J. Electronic Testing* 34, 2 (2018), 183–201.

[11] Farimah Farahmandi, Yuanwen Huang, and Prabhat Mishra. 2017. Trojan localization using symbolic algebra. In *22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. 591–597.

[12] Farimah Farahmandi and Prabhat Mishra. 2019. Automated Test Generation for Debugging Multiple Bugs in Arithmetic Circuits. *IEEE Trans. Computers* 68, 2 (2019), 182–197.

[13] Anna Goldie and Azalia Mirhoseini. 2020. Placement Optimization with Deep Reinforcement Learning. *CoRR* abs/2003.08445 (2020).

[14] Yi Han, Sriharsha Etigowni, Hua Liu, Saman A. Zonouz, and Athina P. Petropulu. 2017. Watch Me, but Don't Touch Me! Contactless Control Flow Monitoring via Electromagnetic Emanations. In *Proceedings of the 2017 ACM*. 1095–1108.

[15] Yuanwen Huang, Swarup Bhunia, and Prabhat Mishra. 2016. MERS: Statistical Test Generation for Side-Channel Analysis based Trojan Detection. In *ACM SIGSAC Conference on Computer and Communications Security*. 130–141.

[16] Yuanwen Huang, Swarup Bhunia, and Prabhat Mishra. 2018. Scalable Test Generation for Trojan Detection Using Side Channel Analysis. *IEEE Trans. Information Forensics and Security* 13, 11 (2018), 2746–2760.

[17] Sami Khairy, Ruslan Shaydulin, Lukasz Cincio, Yuri Alexeev, and Prasanna Balaprakash. 2019. Reinforcement-Learning-Based Variational Quantum Circuits Optimization for Combinatorial Problems. *CoRR* abs/1911.04574 (2019).

[18] Yangdi Lyu and Prabhat Mishra. 2018. A Survey of Side-Channel Attacks on Caches and Countermeasures. *J. Hardware and Systems Security* 2, 1 (2018), 33–50.

[19] Yangdi Lyu and Prabhat Mishra. 2019. Efficient Test Generation for Trojan Detection using Side Channel Analysis. In *Design, Automation & Test in Europe Conference (DATE)*. 408–413.

[20] Yangdi Lyu and Prabhat Mishra. 2020. Automated Test Generation for Trojan Detection using Delay-based Side Channel Analysis. In *Design, Automation & Test in Europe Conference (DATE)*.

[21] M. Pecht and S. Tiku. 2006. Bogus: electronic manufacturing and consumers confront a rising tide of counterfeit electronics. *IEEE Spectrum* 43, 5 (2006), 37–46.

[22] Devendra Rai and John Lach. 2009. Performance of Delay-Based Trojan Detection Techniques under Parameter Variations. In *IEEE International Workshop on Hardware-Oriented Security and Trust, HOST*. 58–65.

[23] Mohammad Tehranipoor and Farinaz Koushanfar. 2010. A Survey of Hardware Trojan Taxonomy and Detection. *IEEE Des. Test Comput.* 27, 1 (2010), 10–25.

[24] John Villasenor and Mark Tehranipoor. 2013. Chop shop electronics. *Spectrum, IEEE* 50 (10 2013), 41–45. https://doi.org/10.1109/MSPEC.2013.6607015

[25] Ronald J. Williams. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.* 8 (1992), 229–256.

[26] Chunyi Wu et al. 2019. Explore Deep Neural Network and Reinforcement Learning to Large-scale Tasks Processing in Big Data. *Int. J. Pattern Recognit. Artif. Intell.* 33, 13 (2019), 1951010:1–1951010:29.