ATPG Binning and SAT-Based Approach to Hardware Trojan Detection for Safety-Critical Systems

Animesh BasakChowdhury[1(✉)], Ansuman Banerjee[2], and Bhargab B. Bhattacharya[2]

[1] Verification and Validation Group, TCS Research, Pune, India
animeshbchowdhury@gmail.com
[2] ACMU, Indian Statistical Institute, Kolkata, India

Abstract. Combating threats and attacks imposed by Hardware Trojans that are stealthily inserted in hardware systems, has surfaced as a challenging problem in recent times. Such threats degrade the reliability and endanger security of the system. Due to scalability issues, Trojan detection remains an extremely difficult problem, especially, when the circuit size is large and Trojan sizes are small. Hardware Trojan is surreptitiously inserted into the design by selecting a few circuit nodes, where rare logic value occurs. This makes their detection probability negligibly small, thereby rendering the arrival of an input combination activating the same, an extremely rare event. Since the number of such Trojans may be exponentially large in terms of such rare nodes, almost all state-of-art techniques suffer from scalability bottlenecks and coverage issues, while generating test vectors. In this work, we propose a systematic approach to sampling in order to lessen the search space, yet preserving the diversity of population. We use binning of trigger-population based on Automatic Test Pattern Generation (ATPG), and invoke Boolean Satisfiability (SAT) solvers to generate test vectors with high Trojan coverage. Simulation results demonstrate the effectiveness and superiority of our method with respect to prior work in terms of Trojan coverage and the cardinality of the test set.

Keywords: Hardware trojan · Activation nodes · Trigger instance
Trojan instance · Trigger · Payload · ATPG binning

1 Introduction

Malicious tampering of hardware designs in a digital system with Trojans and backdoor poses a severe security threat in recent times, endangering the normal functioning of the system quite unexpectedly [21]. Hardware Trojans (HT) are additional circuit elements that are stealthily inserted into the design by adversaries. During functional operation, the design produces the correct behavior most of the time for most of the input patterns; however, when certain input

M. H. Au et al. (Eds.): NSS 2018, LNCS 11058, pp. 391–410, 2018.
https://doi.org/10.1007/978-3-030-02744-5_29

patterns are fed, one or more outputs produce behaviors that deviate from the expected values.

An adversary, who intends to insert Trojans for corrupting a design can attack it at different stages of the System-on-Chip (SoC) life-cycle. In [23], it has been demonstrated that the use of analog malicious hardware built with a capacitor and a few transistors may replace digital counter-based triggers and jeopardize the system. In [3,4,8,20,21], authors have discussed details of the potential stages of the SoC flow, where Trojans may be inserted. In particular, the phases where third-party Intellectual Property (IP) blocks are used or technology mapping by third-party vendors is needed, are more conducive to Hardware Trojan insertion. In general, the aim of Trojan insertion is:

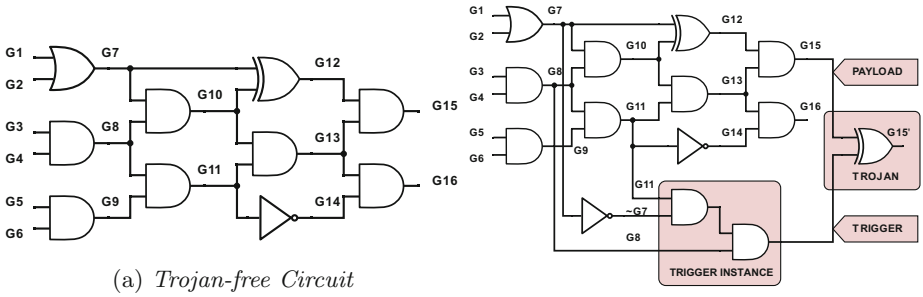
(a) The erroneous functionality ought not to be easily exposed while performing conventional manufacturing testing.
(b) The triggering of the Trojan must be an unusual but valid event.

Modifications and tampering done during the pre-silicon stage for Trojan insertion are outside the scope of this discussion. Here, we are mainly focused on such tampering, that can be non-destructively effected in the post-silicon stage thereby, modifying the netlist, such that the change in functionality can escape the normal ATPG test patterns [21]. Thus, one possible way of inserting Trojans by an adversary is to identify certain states or input combinations in the given circuit, which are extremely *rare*. Whenever any such state (or input combination) is reached, the Trojan is activated and some unexpected behavior is observed at one or more primary outputs. The detection of Trojans heavily relies on how they are modeled and the techniques used to detect them.

In this paper, we explore the problem of test set generation for hardware Trojan detection, using a novel combination of ATPG and SAT. We believe that a disciplined combination of sampling, ATPG and SAT techniques can serve as an effective aid in targeting rare Trojans and detection of their insertion points. We study the shortcomings of the methods in existing literature, and present a novel approach which can generate quality test sets that increase Trojan coverage to a great extent, within reasonable CPU time. We present experimental results to demonstrate the scalability and coverage advantage our method achieves over others. The rest of the paper is organized as follows. Section 2 describes prior approaches used for the detection of hardware Trojans. Section 3 presents our main idea and the methodology proposed to solve this problem. Section 4 presents details of our experimental set-up and results. Section 5 concludes the discussion with notes on possible future directions.

2 Background and Related Work

In this section, we first present an example to illustrate the problem at hand. Figure 1a shows a simple combinational circuit free from Trojans. For node $G7$ to attain the logical value 0, $G1$ and $G2$ both have to be set at 0. Similarly for $G8 = 1$, both $G3$ and $G4$ should be having value 1, and for $G11 = 1$, all $G3$, $G4$, $G5$ and $G6$ should be set at 1.



(a) *Trojan-free Circuit*

(b) *Trojan-affected Circuit*

Fig. 1. Figure showing Trojan-free and Trojan-affected circuit

Therefore, it can be concluded that the probability of occurrence of the logic value 0 at $G7$ is 0.25, i.e. out of every 4 test vectors, there exists only 1 test vector, which can activate the state $G7 = 0$. For $G8 = 1$ and $G11 = 1$, the respective probabilities of occurrence are 0.25 and 0.0625. Hence, the combination $G7 = 0, G8 = 1$ and $G11 = 1$, is expected to occur very rarely. In other words, the state having simultaneous occurrence of these three logic values is relatively uncommon. Only 1 out of 2^6 possible test vectors can drive the circuit to this state (G1 = G2 = 0 and G3 = G4 = G5 = G6 = 1). Such an occurrence can be appropriately termed as a rare event. Even though the combination is rare, it must be a valid activation, and the error must be observable at some outputs. Otherwise, the Trojan will never be triggered, thereby defeating the whole purpose of the adversarial attacker. Hence, there should exist a test input that can expose it. In most of the cases, the trigger's presence can be suitably modeled by a stuck-at fault (s-a-f). The combination of rare nodes provides a favorable location for an adversary where a Trojan instance is likely to be created. In Fig. 1b, we have created a conjunction of three nodes $G7 = 0$, $G8 = 1$ and $G11 = 1$, where the occurrence of logic 1, is a rare event. Under a particular input combination, the output of the resultant conjunction will become high and corrupt the logic value at the output node $G15$. We first define a few terminologies used frequently throughout this work.

Rareness Threshold: Using structural analysis of a given netlist, the probability of occurrence of logic values 0 and 1 at a circuit node can be determined statically. A *rareness threshold* value is set to determine the rarity of occurrence of a logic value at a particular circuit node. We denote this rareness threshold by θ.

Activation Node: A node where the probability of occurrence of a logic value is less than the *rareness threshold* can play a dominant role in creation of a trigger instance, which in turn can serve as an *activation node* for a Trojan in the netlist.

Trigger Instance: An instance formed by the conjunction of several *activation nodes* is termed a *trigger instance*. The output of the conjunction (AND gate) is the *trigger*.

Payload: A node in the netlist, whose logic value is corrupted by the trigger is a *payload*. Any internal node or a primary output is a fertile ground for such logic corruption.

Trojan Instance: A suitable trigger-payload combination is termed as a *Trojan instance*. Typically, the output of the trigger is XOR-ed with the payload to flip the expected logic value at that point, and such a modification creates a Trojan.

Q-Value: Number of activation nodes used to create a trigger instance, denoted by Q. This value determines the size of the trigger.

Controllability: Controllability of a trigger instance is a measure of hardness of trigger activation.

Observability: Observability can be defined as how hard/easy is the propagation of corrupted logic value occuring at a payload, at any Primary Output (PO).

A low-controllable low-observable trigger-payload combination, constitutes an ideal Trojan. The main challenge arises when an adversary chooses the activation nodes by setting a very small value to θ; as a sequel, a combination (ANDing) of such nodes would become extremely rare to happen. Trojan detection has been an active area of research in recent times in the hardware security research community [2,5,6,10,15,17,22]. Significant work has been done in the field of hardware Trojans, focusing mainly on the insertion strategy and techniques for detecting them. These include, among others, observing side-channel parameters such as power surge, delay analysis, and path propagation for detecting the presence of Trojans [1,9,16]. However, most of these techniques fail to detect the Trojans, especially when there are non-uniform variations of side-channel parameters of the golden design in the design-under-test (DUT). Test-based approaches have also been extensively studied and the derivation of the MERO test pattern is an important research contribution in this direction [5]. Other techniques such as ODETTE [2], DFTT [10] and TeSR [15], are also capable of producing efficient test patterns for detecting Trojans. The authors in [17] proposed an improved version of MERO aiming to maximize Trojan coverage. Several other techniques have focused on the prevention of hardware Trojan attacks on a given design [6,22]. In a very recent work [7], authors have proposed a hybrid technique of using model checking and ATPG for Trojan detection. However, the threat model is not very generic. They have considered the output of non scan flip flops (FF) in a partial scan sequential design, as the point of attack. In our paper, we have used the same threat model as used in MERO and the improved MERO models, and hence is completely different from the one proposed in [7]. Therefore, test generation, in particular, for the threat model used in [7] is beyond the scope of our work. Nevertheless, our test generation methodology on full scan sequential circuits is an over-approximated modeling of partial scan designs and can uncover the Trojan set, based on the threat model proposed for partial scan design.

In this paper, we focus mainly on the development of a scalable test generation framework for uncovering covertly inserted Trojans. Since the number of possible Trojan instances can be exponential, determining a test set with substantial Trojan coverage suffers from severe scalability issues when circuit size is large. The authors in MERO [5] and Improved MERO [17], have described statistical and heuristic techniques for generating test pattern to detect hardware Trojans. In MERO, an N-detect test set is generated for the activation nodes. Effectively, from a set of random test patterns, test vectors are applied in an iterative manner so that each activation node is excited to the rare logic value at least N times. Increasing N effectively increase the probability of trigger activation. As the primary focus is on small sized Trojan instances, the maximum number of activation nodes that can participate in a trigger instance has been considered upto four [4,17]. In the improved MERO version, the authors have used genetic algorithm (GA) combined with payload-aware test generation and a SAT-based technique for detecting extremely hard-to-detect Trojan instances. Although the results outperform the MERO-based approach, the framework samples the trigger instances randomly only once from entire population. There are certain open issues that need to be addressed:

(a) MERO [5], and Improved MERO [17] are evaluated on the ISCAS85 and ISCAS89 circuits only. In MERO, random 100k test patterns are selected initially to derive optimized test patterns. In improved MERO, initially 100k trigger samples for ISCAS85 circuits, and 10k instances for ISCAS89 circuits are chosen randomly to generate test vectors. But, when circuit size increases, one time random sampling may result in poor sampled set of trigger population. There is a high chance that the sampled population does not contain all activation nodes, as part of trigger instances.
(b) MERO [5] is a lightweight heuristic framework, which takes care of controllability of trigger instances. It does not consider the observability of malfunctioned logic at the output. In improved MERO [17], payload aware test vector has been generated. However the process initially optimizes trigger coverage, and then based on the test vectors generated, pseudo test vectors (PTV) are created to cover feasible payloads. Thus, payload aware test vector generation gives priority to maximizing trigger coverage over feasible payload coverage.

Motivated by the above limitations, our main aim is to derive an efficient and high coverage test-set for detecting Trojans. The detection process has to be fast and efficient, and take less CPU-time. We consider hardware Trojan instances that are non-destructively inserted in a logic circuit during the post-silicon phase utilizing the rarity of activation nodes. To address the coverage problem, we introduce a well organized, judicious and cost effective sampling of trigger population. The modified sampling technique ensures that the sampled population contains all activation nodes in right proportion. The initial population plays a major role, in determining the quality of test vectors generated. More the variety amongst trigger instances in population, greater is the heterogeneity of test vectors. Again, for scaling up the process, we use a

divide-and-conquer strategy called ATPG-binning, which helps in partitioning the population into disjoint sets and solve a larger problem, by solving multiple sub-problems. We then generate test patterns using classical ATPG tools for normal trigger instances and SAT-based techniques for hard-to-detect trigger instances. SAT-based methods are computationally costly, taking one trigger at a time. Owing to recent advances in SAT-solving techniques, infeasibility test of trigger activation can be done in reasonable time. However, there can still exist very few cases, where SAT-solver is unable to generate result within a specified time limit. In such cases, we call the trigger as unsolvable. Hence, it is a reasonable choice to employ the SAT tool on trigger instances declared aborted by structural ATPG tools. This three step methodology is able to detect a large number of Trojan instances within a reasonable CPU-time on large sized circuits. Our method provides a general framework for Trojan analysis that can be used to warrant certain level of trust and reliability of safety critical applications. The main contributions of this paper are as follows:

(1) A simple sampling technique to choose an initial trigger population over which the test vectors are ought to be generated. This sampling technique guarantees the presence of each activation node in right propotion and more heterogeneity.
(2) A scalable methodology of test vector generation in the form of ATPG binning, which takes as input a sampled trigger population. Sampling ensures that the population is qualitatively and quantitatively good, to generate high quality test vectors. To scale up the framework on a large trigger population, a divide and conquer strategy called ATPG binning is employed.
(3) The framework tries to improve the trigger and Trojan coverage simultaneously. Iteratively, for a given set of test vectors, trigger and payload coverage are computed, and a subset of them is chosen, ensuring high coverage of both parameters. This is done in both steps, during ATPG binning and SAT.

3 Methodology

We now present a detailed methodology of test vector generation for Hardware Trojan detection. Our discussion has been broadly divided into three main sections:

1. Static analysis and initial sampling of trigger population.
2. Test generation using ATPG binning.
3. SAT-based test generation for hard-to-detect triggers.

3.1 Static Analysis and Initial Sampling

Initially, we analyze the circuit statically, using structural analysis. For proper identification of suitable trigger candidates, we determine the probability of occurrence of logic values, 0 (P_{zero}), and 1 (P_{one}), for each node. Figure 2 shows

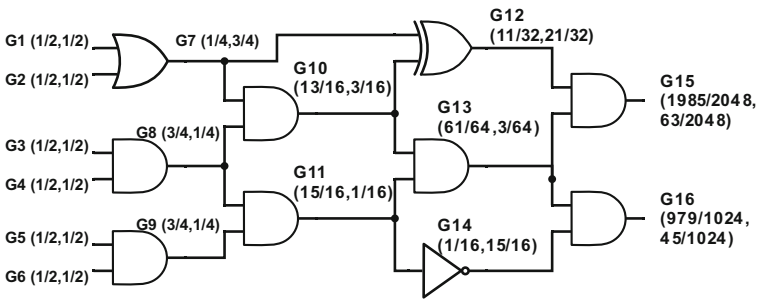


Fig. 2. The probabilities of a line being at logic 0 and 1 are shown as ($Prob_0$, $Prob_1$)

the probability values of logic 0 and 1 for all nodes in a typical netlist. We adopted the framework given in [19] to determine rarity of logic values at nodes. The rareness value has to be properly defined at the user end. Based on the rarity value, rareness threshold θ, is set, which roughly furnishes a measure of Trojan stealthiness. Suppose, Attacker 1 chooses $\theta_1 = 0.1$ and Attacker 2 chooses $\theta_2 = 0.4$, w.r.t the netlist shown in Fig. 2. For Attacker 1, creating a trigger with a combination of four such nodes would have activation probability value in the range $[(0.1)^4, 0.1]$. Whereas, for Attacker 2, the activation probability is quite high, i.e. $[(0.4)^4, 0.4]$. Higher the activation probability values, higher are the chances of Trojan getting exposed during regular test application. The detailed mathematical proof regarding the probability values associated with the activation of trigger and Trojan instances, have been shown in [5].

Once the design is statically analyzed, all nodes having either P_{zero} or P_{one} value less than θ, are identified. These nodes are called as *activation nodes*, and constitute candidates of trigger instances. We now define the parameters associated with trigger population and activation nodes.

$(\mathbf{i}, \mathbf{L_i})$: The tuple is a representation of an activation node and its associated rare logic value. i denotes the node name, and L_i denotes the rare logic value.

Activation Node Set R: Set of all activation nodes in a given netlist N. $R = \{(i,L_i), \forall i \in N$ and $P(L_i) < \theta\}$.

In literature, it is already established that a Trojan created from a trigger instance having a large Q-Value, is easily detectable with side channel analysis. But, when the trigger-size is small ($Q < 5$), the false positive rate is alarmingly high. Therefore, testing based approaches are tightly coupled with side channel analysis techniques to uncover more trojans. We proceed with our methodology, keeping our focus on small sized Trojans. Accordingly as in literature, we restrict our choice of Q-Value up to 4.

Even though the Q-Value is low, the search space of trigger instances is exponential in terms of the number of *activation nodes* of a circuit. For Trojan instances, it is again multiplied by another exponential factor of number of possible payloads. Hence, it is practically infeasible to generate test vectors covering the entire population. This demands the necessity of a quality sampled trigger

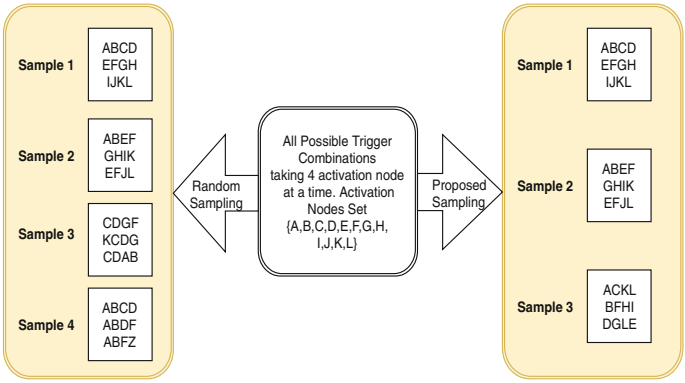


Fig. 3. Sampled sets of trigger using random sampling and proposed sampling.

population, which can ensure decent coverage over entire search space. Rather than going for random sampling, we propose a novel strategy of sampling trigger instances to generate a quality initial sampled set. We now describe, how and why our modified sampling technique is capable of generating good quality test vectors.

In Fig. 3, we have presented an example of the modified sampling approach. Initially, we have a set of 12 distinct activation nodes. Taking Q-Value $= 4$, there can be $\binom{12}{4}$ possible trigger instances. Now, we have the option to take all $\binom{12}{4}$ trigger instances as initial population for test vector generation. But, as circuit size increase, the number of activation nodes increases dramatically. Hence, intelligent sampling is required out of $\binom{12}{4}$, for scalability purpose. MERO [5] and Improved MERO [17] use random sampling of fixed 100k instances. For the sake of explanation, consider we are interested to sample three trigger instances out of $\binom{12}{4}$ possible combinations. Now, if we go for random sampling, each time we may end up having different samples. As shown in Fig. 3, there are 4 different samples of trigger population that we can possibly get from random sampling. After a careful observation, one can conclude that both Sample 1 and Sample 2 are quite good. They contain all 12 activation nodes distributed over 3 trigger instances, whereas, Sample 3 and Sample 4 do not cover all of them. Therefore, test vectors that are derived by ATPG targeting Sample 1 and Sample 2, would certainly yield superior coverage over those needed for testing Sample 3 and Sample 4. Upon close examination, it can be seen that test vectors generated for Sample 3 will be biased towards activation nodes A and B. Similarly, for Sample 4, it would be biased towards nodes C and D. Hence, with a very high probability, test vectors would miss any trigger created with an activation node, missing from the sampled set. In order to get quality test vectors, it is necessary that the initial sample population of trigger instances contain enough information, diversity and lesser correlation amongst trigger instances. We now introduce our modified sampling approach based on the Q-Value and the activation set R. We first define some parameters used in our sampling approach.

numCountInSample$_X$: Count of the Activation node X present in trigger instances of Sampled Trigger Population.

popCount: Minimum threshold count to be maintained for each activation node present in trigger instances of the sampled trigger population.

Initially, for all activation nodes, the numCountInSample value is initialized to 0. In the first iteration of sampling, a set *chooseSet* is initialized to R. Now, *Qvalue* number of activation nodes are chosen from the *chooseSet* without replacement. The trigger crafted out of that, is checked whether it is already present in the Sampled Trigger Population (T). If it is present, the trigger is discarded. The process of creating new trigger continues, until the new trigger is absent from T. Then, the trigger is included in T, only if there exists an activation node a for which $numCountInSample_a < popCount$. Once the trigger is included, the *numCountInSample* value for all activation node in the trigger, is incremented by 1. Trigger creation using activation nodes from *chooseSet* continues, until the cardinality of *chooseSet* becomes less than $QValue$. After that, the *chooseSet* is again reinitialized to R, and the process is repeated. Note that, in one iteration, triggers created will not have any activation node in common. The sampling process completes when *numCountInSample* for each activation node, is at least popCount. In Fig. 3, we have applied our sampling technique and generated the sampled trigger population, keeping popCount = 1. Such a sample automatically guarantees better test vector generation as compared to random samples. Algorithm 1 presents the relevant steps related to static analysis and the sampling mechanism.

3.2 ATPG Binning

In our methodology, our sampling criteria ensures that quality test-sets are generated. For generating test vectors for a trigger instance, we model trigger activation as a single stuck-at fault (s-a-f). We apply stuck-at 1 (s-a-1) fault at the output of the trigger, and go on to generate test vectors for the same. As the number of elements in the sampled trigger population is high, it is practically infeasible for a structural ATPG tool to generate them in one go. To make our method scalable, we divide the population into smaller disjoint bins randomly. The set, containing all the bins, is called K. The number of trigger instances in a bin depends on the maximum number of primary outputs (POs) that an ATPG tool can handle. This step ensures that we generate efficient test vectors without hitting the scalability bottleneck.

A typical modified netlist consists of all trigger instances of the bin, additional inserted as POs, in the original netlist. Now, for each modified netlist (corresponding to each bin) in set K, a structural ATPG tool is deployed. We use Deterministic Test Pattern Generation (DTPG) in our test vector generation approach using structural ATPG. As a result, testcubes are generated, consisting of $X(Don't\ Care)$ terms. The output reports presence of three kinds of trigger instances: (a) *Feasible Trigger Instances*, for which test cubes have been generated, (b) *Redundant Trigger Instances*, i.e. no test vector exists to activate the trigger instance (can be safely ignored as infeasible triggers), and (c)

Algorithm 1. Creation of initial sampled trigger population

Input:
N : Gate level netlist
θ : Rareness threshold
q : Q-Value
popCount : Minimum threshold count of activation node to be present in trigger instances of Sampled Population.

Output: T : Sampled trigger population.

Read gate level netlist of design.
for $\forall$ node $i \in N$ **do**
  Calculate $P_{\text{zero}}(i)$ and $P_{\text{one}}(i)$
  if $P_{\text{zero}}(i) < \theta$ **then**
    $R \leftarrow R \cup (i, 0)$
  else
    if $P_{\text{one}}(i) < \theta$ **then**
      $R \leftarrow R \cup (i, 1)$
    end if
  end if
end for
Set R is reported as set of tuples (i,L_i), where i is the node and L_i is its associated Rare logic value.
Initialize $T \leftarrow \phi$.
for *each* activation node $i \in R$ **do**
  Initialize $numCountInSample_\text{i} \leftarrow 0$
end for
while $\exists$ activation node $i \in R$, s.t. $numCountInSample_\text{i} < popCount$ **do**
  Initialize $chooseSet \leftarrow R$.
  while $|chooseSet| >= q$ **do**
    Generate a trigger instance $TRIG$, choosing q tuples from chooseSet, without replacement.
    if $TRIG \notin T$ **then**
      if $\exists$ activation node $i \in TRIG$, s. t.$numCountInSample_\text{i} < popCount$ **then**
        $T \leftarrow T \cup TRIG$
        Increment $numCountInSample$ by 1, $\forall$ activation node $i \in TRIG$.
      end if
    end if
  end while
end while
Report set T.

Aborted Trigger Instances, which can be categorized as extremely hard and rare trigger instances. Structural ATPG tool failed to report whether such instances are feasible or not. After testcubes generation, we apply a lightweight compaction methodology as described in [14] for testcube compaction. We cluster the testcubes based on similarity of skeleton structure, and then construct the parent testcube set T_x, having four logic values 0, 1, X (Don't care) and

C (Contradict). This compaction is done in order to preserve important test cubes. The test vector generated from these test cubes would be taken into the final Trojan detection test-set, based on the coverage efficiency. Let us define two types of coverages, which we have used to determine the quality of a test vector, in the rest of the paper.

Trigger Coverage of a test vector is defined as the number of trigger instances, which can be activated by the application of the test vector.

Stuck-at fault Coverage of a test vector is defined as the number of nodes in the circuit, whose stuck-at fault can be detected by the test vector.

We now present **Theorem** 1, which basically relates trigger coverage and overall *s-a-f* coverage of a test vector with Trojan coverage.

Theorem 1. *Let G be an internal node, which has been XORed by a trigger instance T. If there is a test vector t_i that activates trigger T, and detects a s-a-f at G, then t_i will be able to detect the Trojan consisting of T as trigger and G as payload.*

An illustration of this result is shown in Fig. 4.

As a consequence of Theorem 1, we can expand the parent testcube set T_x, and calculate the trigger Coverage and stuck-at fault coverage of the generated test vector. Test vectors providing coverage of unique triggers and stuck-at faults are taken into the master test-set TA. Now, for each parent testcube in the compressed testcube set T_x, test vectors are generated by randomly filling up with 0s and 1s at X bit. For every C (Contradict) bit of the test cube, a pair of test vectors should be generated, one having 0, other having 1, as shown in Fig. 5. Now, for each test vector, we check for trigger coverage in the bin, and s-a-f coverage in the entire circuit. The test vector is included in the master testset TA, if any new trigger is covered from the bin or the overall *s-a-f* coverage of the circuit is increased. After all feasible trigger instances from the bin are covered, the next p consecutive steps are checked to see if there is any increase in *s-a-f* coverage. If there is no increase in *s-a-f* coverage, we stop adding test vectors to the master test set TA. The current bin is called *explored*, and removed from set K.

Once a bin is explored, for all the test vectors in TA, we check for trigger coverage of bins already present in K. The triggers, which are covered by TA, are appropriately removed from their respective bins. Now, the bins are arranged in decreasing order, according to the number of triggers present. The bin, having highest number of triggers, is taken into consideration. The process continues until all the bins are covered. After all the steps are performed, we get the master testset TA, and the set of aborted trigger instances of all the bins. Figure 6 shows the overall workflow of our algorithm.

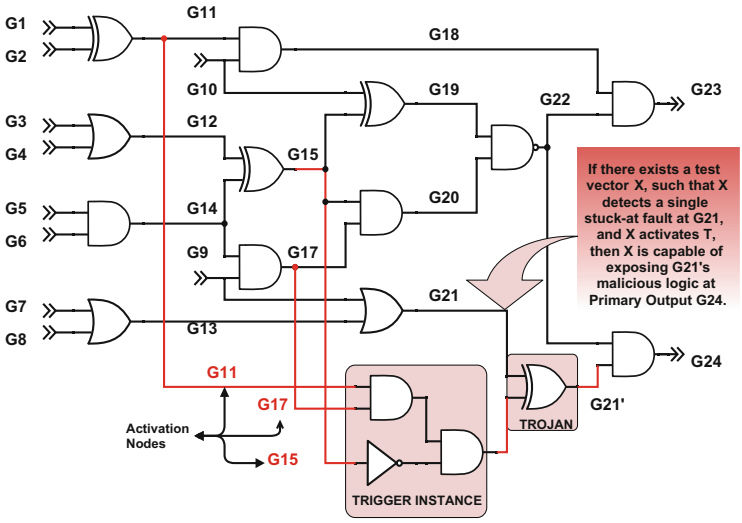


Fig. 4. A test vector covering the single s-a-f at internal node G21 and trigger T can uncover the Trojan created by the combination of G21 and T.

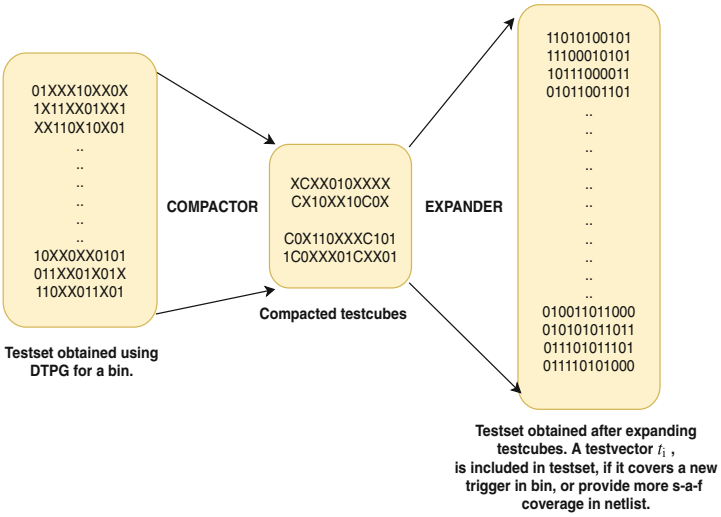


Fig. 5. Compaction and expansion of test cubes for each bin under consideration

3.3 SAT Methodology

SAT-based test generation methodology has been used in recent times, especially to report test vectors for hard-to-detect faults. A test generation problem can be suitably converted into a Boolean Satisfiability problem. Owing to efficiency of

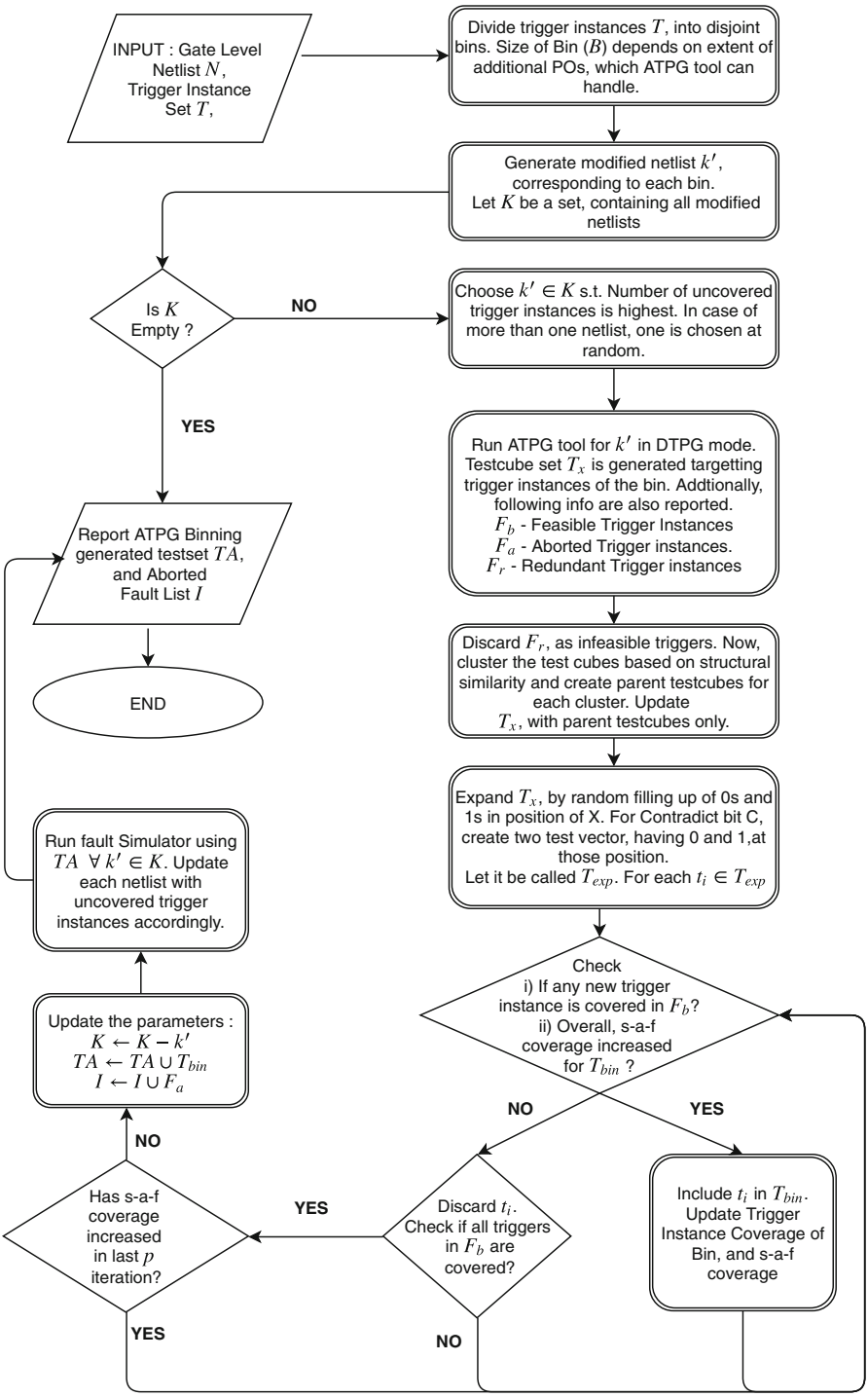


Fig. 6. Flowchart of ATPG binning algorithm

powerful SAT-solvers, we use this methodology to report test vectors for *aborted trigger instances* reported from the structural ATPG tool.

Using Tseytin transformation, the gate level netlist is converted to Conjunctive Normal Form. For each aborted trigger instance, CNF clauses are created conjuncting the clauses of the original circuit and the clause representing the trigger. The clauses are then fed to a SAT-solver to check for existence of any satisfiable assignment. Modern SAT-solvers are able to precisely arrive at two distinct decisions - SAT and UNSAT, in a reasonable time. SAT implies that the trigger is feasible. The input test vector can be fetched from the instance returned from SAT solver. UNSAT denotes no test vector exists for the trigger.

For increasing the s-a-f coverage, we try to generate M distinct satisfying instances for each aborted trigger instance, provided it is feasible. We first compute the *s-a-f* coverage of all M test vectors. Out of M, we choose a subset S_M. Figure 7 shows the coverage of M test vectors, for a typical aborted trigger instance. The subset S_M is chosen such that it provides same overall *s-a-f* coverage, as that of M vectors. We employ a simple greedy algorithm for creating S_M. Initially, we put each test vector into S_M, which covers a unique *s-a-f* of a node. Once it is done, overall coverage by S_M is computed, and checked to see if there exists any s-a-f not covered by S_M, but still coverable by M. Then, for each s-a-f still uncovered, we pick a random test vector covering it, and overall s-a-f coverage is computed again, including that test vector into S_M. This process continues, till s-a-f coverage of S_M is equal to the s-a-f coverage of M. S_M is assigned as M when each of the M test vectors provide distinct coverage. We repeat this step for each of the aborted instances. The triggers reported as UNSAT from the SAT solver can be regarded as infeasible trigger instances, and hence can be neglected. At the end, test vectors generated from SAT and those from structural ATPG are combined to report the final testset for Trojan detection. The detailed flow is shown in Algorithm 2. This 2-step methodology used for obtaining the testset ensures high confidence level of coverage of all feasible triggers and Trojans, given the value of q and θ.

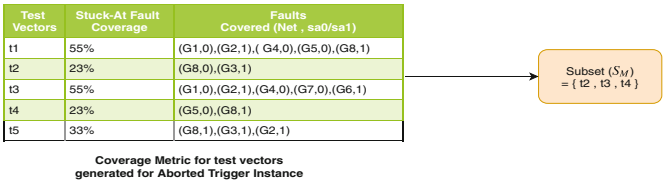


Fig. 7. Coverage for test vectors generated by SAT-solver for a typical trigger instance. S_M denotes the subset of test vectors which provide maximum coverage.

Algorithm 2. Test generation with SAT-solver

Input:

N : Gate level netlist
TA : Testset TA generated by ATPG binning
I : Aborted trigger instance set
M : Number of distinct test vectors required for each aborted trigger to find efficient test subset.

Output: $TEST_{\text{final}}$: Combined testset - Testset generated from SAT ($TEST_{\text{SAT}}$) + Testset generated from ATPG Binning (TA).

Read netlist N, do Tseytin transformation and generate CNF of netlist, C.
Initialize $TEST_{\text{SAT}} \leftarrow \phi$.
for each trigger instance $t \in I$ **do**
    Generate CNF for trigger instance t, C'.
    $C' \leftarrow C \bigwedge C'$
    Initialize $numTestVecGenerated \leftarrow 0$.
    $testSetForTrig \leftarrow \phi$
    while $numTestVecGenerated \neq M$ **do**
        Check for satisfiable instance for C'.
        if Instance is SAT **then**
            Retrieve test vector, t_{SAT}.
            $testSetForTrig \leftarrow testSetForTrig \cup t_{\text{SAT}}$.
            $C' \leftarrow C' \bigwedge \sim t_{\text{SAT}}$.
            Increment $numTestVecGenerated$ by 1.
        else
            Instance is UNSAT.
            break
        end if
    end while
    if $numTestVecGenerated == 0$ **then**
        Trigger instance is infeasible.
    else
        Generate 's-a-f' coverage metric for each test vector $\in testSetForTrig$.
        Choose a subset S_M from testSetForTrig, which maximise the overall 's-a-f' coverage.
        $TEST_{\text{SAT}} \leftarrow TEST_{\text{SAT}} \cup S_M$.
    end if
end for
$TEST_{\text{final}} \leftarrow TEST_{\text{SAT}} \cup TA$
Report testset $TEST_{\text{final}}$.

4 Experimental Results

We carried out simulation on combinational benchmarks from ISCAS-85, and sequential benchmarks from ISCAS-89 and ITC-99. All sequential circuits are taken in full scan mode. The framework is developed in C++ and Python and the experiment has been carried out on a Linux Workstation with Intel Xeon E5 3 GHz Processor and 32 GB RAM. We used the Transition Probability Calculation (TPC) [18] tool from trust-hub.org. The tool takes a circuit netlist as

input, statically analyses the structure and reports the probability of occurrence of 0 (P_{zero}) and 1 (P_{one}) for circuit nodes. We next selected the value of the rareness threshold. For the sake of comparison with previous state-of-art techniques MERO [5], Improved MERO [17], we took $\theta = 0.1$ (for ISCAS-85 circuits) and 0.01 for full scan ISCAS-89 and ITC-99 circuits respectively. The Q-Value was taken as four, and $popCount = 5000$ for our experimental results. The values of Q-Value, θ, and $popCount$ taken in this set-up, ensure the test generation is targeted for those Trojans, which are rare and remain hidden during normal ATPG test. An appropriate $popCount$ value is taken to ensure sufficient presence of all activation nodes in the sampled population. However, increasing $popCount$ value is going to increase initial population. Depending on the need, the parameter can be suitably tuned to get enough diversity. The set of activation nodes is fed to program *genTrojanComb*, that generates sampled trigger population. The sampled population is then divided into several disjoint bins by dumping them into individual files. The Bin Size B, is determined by the limit of the ATPG tool. In order to harbour more POs, we modified the source code of the ATALANTA ATPG tool [11] to process maximum number of POs in a single run. The original netlist along with a bin are provided as input to the program *TrojanInjection*, that outputs the modified netlist. The process is repeated for all the bins and the output generated at the end contains a set of modified netlists K. Thereafter, the ATPG Binning Algorithm is deployed on the set K for end-to-end run using structural ATPG ATALANTA, compactor [14], expander and HOPE fault simulator [12]. At the end, we get testset TA, aborted trigger instances I and redundant trigger instances. All redundant triggers can be safely ignored as infeasible/false trigger instances. Aborted trigger instances can be considered as hard-to-trigger instances, owing to failure of test vector generation in a stipulated time. We used the SAT-solver to generate test vectors, for the triggers present in the aborted fault list I. For each trigger in I, the instance, along with the original netlist is fed to a *createSATInstance* tool. The SAT-formulation for the trigger instance is then fed to the SAT-solver, *zChaff* [13]. A Python wrapper has been used over *zChaff*, to produce M distinct input vectors for a trigger instance. In order to produce distinct test vectors in each iteration, the test vector t_{SAT} generated in the current iteration is negated and then conjuncted with existing CNF. This ensures the same test vector is not generated twice. To maintain the trade-off between computation time, number of test vectors, and *s-a-f* coverage over the netlist, we take value of $M = 5$. Once the test vectors are generated for a trigger, the coverage is computed and S_M is determined. The iteration continues till all the aborted triggers are covered. The test vectors generated by the SAT-solver are then combined with testset TA, to report the final test vector set.

The results in Tables 1 and 2 show the effectiveness of the proposed method over existing test based approaches for Trojan detection. Table 1 presents the efficacy of the approach on standard ISCAS85 and ISCAS89 benchmarks and large industrial benchmarks like ITC99. To the best of our knowledge, most of the techniques have considered only 0.1 million sampled trigger instances, with no information about quality of initial population. Our technique provides 100%

Table 1. Table showing test vectors generated by the proposed scheme. θ is 0.1 for ISCAS85 combinational circuits and 0.01 for ISCAS89 and ITC99 benchmark circuits. popCount = 5000 for initial sampled trigger population.

Benchmark circuits	No. of activation nodes	Trigger instances in sampled population	Feasible trigger instances	Testset length			CPU Time (in seconds)
				Testset generated by ATPG binning	Testset generated by SAT	Total testset generated	
c432	40	57387	56181	534	0	534	0.5
c499	48	71893	4764	1421	15	1436	23.1
c880	62	91964	86192	2097	71	2168	94.2
c1355	112	167231	1432	1876	0	1876	110.3
c1908	65	89267	82141	3387	2967	6534	3005.7
c2670	67	97129	92110	4329	1108	5437	1478.5
c3540	196	261152	193475	4126	2307	6433	9761.2
c5315	176	237084	221885	9029	5467	14496	22721.3
c7552	232	310872	289116	12674	32101	44775	57643.9
s15850	748	1002785	561038	7824	1583	9407	17298.1
s38417	1254	1622398	1209345	38762	10976	49738	68012.5
b14	711	1012654	632093	18943	6584	25527	45019.4
b15	684	1021997	731092	21721	7894	29615	52310.8
b17	879	1255612	910901	25892	9197	35089	57119.6
b20	970	1474958	877213	19373	10176	29549	84081.4
b21	1472	2040812	1484708	27174	9178	36352	77210.6
b22	1736	2480198	1806734	32023	37434	69457	96211.7

Table 2. Table showing trigger and Trojan coverage with proposed scheme. Trojan sample size - 100K for all ISCAS85, ISCAS89 and ITC99 benchmarks.

Benchmark circuits	Trigger coverage (%)	Trojan coverage (%)	Benchmark circuits	Trigger coverage (%)	Trojan coverage (%)
c432	100	93.12	c7552	79.5	69.51
c499	100	94.1	s15850	77.67	59.09
c880	100	91.87	s38417	71.42	52.8
c1355	99.31	83.67	b14	81.41	63.53
c1908	100	92.3	b17	70.28	60.58
c2670	100	89.1	b20	61.21	52.21
c3540	94.67	78.2	b21	55.78	43.71
c5315	92.81	76.7	b22	53.47	44.1

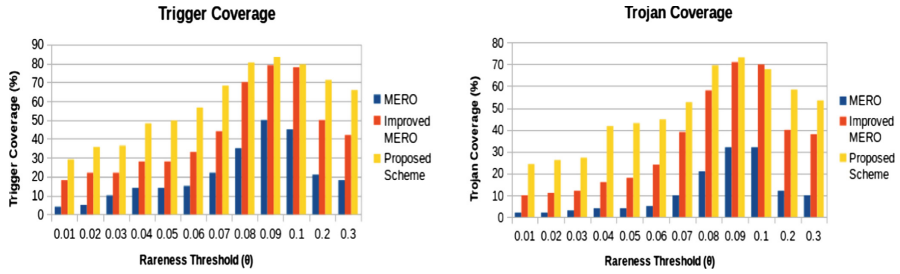
trigger coverage for most of the ISCAS85 circuits. Table 2 shows the coverage of trigger and Trojan instances, over various benchmarks. The trigger and Trojan instances over which coverage results have been shown here, are not part of

Table 3. Table showing comparison of trigger and Trojan coverage for MERO, Improved MERO, and the proposed scheme. For MERO, N = 1000. $\theta = 0.1$ (combinational), 0.01 (sequential). Trojan sample size - 100K (combinational), 10K (sequential)

Benchmark circuits	MERO		Improved MERO		Proposed scheme	
	Trigger coverage	Trojan coverage	Trigger coverage	Trojan coverage	Trigger coverage	Trojan coverage
c880	75.92	69.96	96.19	85.70	100	91.87
c2670	62.66	49.51	87.15	75.82	100	89.1
c3540	55.02	23.95	81.55	60.00	94.67	78.2
c5315	43.50	39.01	85.91	71.13	92.81	76.7
c7552	45.07	31.90	77.94	69.88	79.5	69.51
s15850	36.00	18.91	68.18	57.30	79.21	65.18
s38417	21.07	14.41	56.95	38.10	74.61	58.9

the initial Sampled Trigger Population. Our approach shows better coverage when compared to all previous state-of-art techniques, in terms of trigger and Trojan coverage, even when the circuit size increases considerably. Table 3 shows comparative results of our method with MERO [5] and Improved MERO [17]. Both the techniques suffer from poor coverage when circuit size increases.

We now present a comparative analysis for the circuit c7552, for the proposed method versus state-of-art techniques [5,17]. In Fig. 8a, it is clearly visible that for every value of θ selected, the proposed scheme provides considerable coverage than both the previous techniques. Even when θ is lowered, our scheme is able to uncover triggers almost as twice as the improved MERO version. This is because our approach considers a good initial sample trigger population. It also makes sure of the fact that test vectors are not biased towards certain activation nodes, and trigger population is heterogeneous. In Fig. 8b, Trojan coverage has been compared with already existing techniques and the proposed scheme, over a range of rareness threshold θ values. It is noticeable that even for Trojan coverage, the proposed scheme outperforms the existing techniques.



(a) *Trigger coverage* (b) *Trojan coverage*

Fig. 8. Trigger and Trojan coverage chart for c7552

5 Conclusion

In this work, we have presented a scalable approach for producing an efficient test set that is capable of detecting stealthily-inserted hardware Trojans in a digital circuit. Our method uses a judicious sampling process followed by ATPG-binning that helps to reduce the complexity of the search process significantly, followed by SAT-solving for hard trigger instances. The proposed method can take care of both trigger coverage and feasible payload coverage simultaneously, in order to improve Trojan coverage significantly. Our technique provides a scalable framework for hardware Trojan detection over a generic threat model.

References

1. Agrawal, D., et al.: Trojan detection using IC fingerprinting. In: IEEE S&P (2007)
2. Banga, M., et al.: ODETTE: a non-scan design-for-test methodology for trojan detection in ICs. In: HOST (2011)
3. Beaumont, M., et al.: Hardware trojans-prevention, detection, countermeasures (a literature review). Technical report, DTIC Document (2011)
4. Chakraborty, R.S., et al.: Hardware trojan: threats and emerging solutions. In: HLDVT (2009)
5. Chakraborty, R.S., et al.: MERO: a statistical approach for hardware trojan detection. In: CHES (2009)
6. Chakraborty, R.S., et al.: Security against hardware trojan through a novel application of design obfuscation. In: ICCAD (2009)
7. Cruz, J., et al.: Hardware trojan detection using ATPG and model checking. In: VLSI Design (2018)
8. Jacob, N., et al.: Hardware trojans: current challenges and approaches. IET Comput. Dig. Tech. **8**, 264–273 (2014)
9. Jin, Y., et al.: Hardware trojan detection using path delay fingerprint. In: HOST (2008)
10. Jin, Y., et al.: DFTT: Design for trojan test. In: ICECS (2010)
11. Lee, H., et al.: ATALANTA: an Efficient ATPG for Combinational Circuits. Virginia Polytechnic Institute and State University, Blacksburg (1993)
12. Lee, H.K., et al.: HOPE: an efficient parallel fault simulator for synchronous sequential circuits. In: IEEE TCAD (1996)
13. Mahajan, Y.S., et al.: Zchaff: an efficient SAT solver. In: Theory and Applications of Satisfiability Testing (2004)
14. Mrugalski, G., et al.: Compression based on deterministic vector clustering of incompatible test cubes. In: ITC (2009)
15. Narasimhan, S., et al.: TeSR: a robust temporal self-referencing approach for hardware trojan detection. In: HOST (2011)
16. Rad, R., et al.: A sensitivity analysis of power signal methods for detecting hardware trojans under real process and environmental conditions. In: IEEE TVLSI (2010)
17. Saha, S., et al.: Improved test pattern generation for hardware trojan detection using genetic algorithm and boolean satisfiability. In: CHES (2015)
18. Salmani, H.: TPC: Transition probability calculation (2011). https://www.trust-hub.org/

19. Salmani, H., et al.: A novel technique for improving hardware trojan detection and reducing trojan activation time. In: IEEE TVLSI (2012)
20. Tehranipoor, M., et al.: Trustworthy hardware: trojan detection and design-for-trust challenges. Computer **44**, 66–74 (2010)
21. Xiao, K., et al.: Hardware trojans: lessons learned after one decade of research. In: ACM TODAES (2016)
22. Xiao, K., et al.: A novel built-in self-authentication technique to prevent inserting hardware trojans. In: IEEE TCAD (2014)
23. Yang, K., et al.: A2: analog malicious hardware. In: IEEE S&P (2016)