TroLL: Exploiting Structural Similarities between Logic Locking and Hardware Trojans

Yuntao Liu[1] , *Member, IEEE*, Aruna Jayasena[2] , *Member, IEEE*,
Prabhat Mishra[3] , *Fellow, IEEE*, Ankur Srivastava[4] , *Fellow, IEEE*
[1]*Department of Electrical and Computer Engineering, Lehigh University*
[2]*Department of Computer Science and Engineering, University of Tennessee at Chattanooga*
[3]*Department of Electrical and Computer Engineering, University of Florida*
[4]*Department of Electrical and Computer Engineering & Institute for Systems Research, University of Maryland, College Park*
[1]yule24@lehigh.edu, [2]aruna@tennessee.edu, [3]prabhat@ufl.edu, [4]ankurs@umd.edu

***Abstract*—Logic locking and hardware Trojans are two fields in hardware security that have been mostly developed independently from each other. In this paper, we identify the relationship between these two fields. We find that a common structure that exists in many logic locking techniques has desirable properties of hardware Trojans (HWT). We then construct a novel type of HWT, called Trojans based on Logic Locking (TroLL), in a way that can evade state-of-the-art ATPG-based HWT detection techniques. In an effort to detect TroLL, we propose customization of existing state-of-the-art ATPG-based HWT detection approaches as well as adapting the SAT-based attacks on logic locking to HWT detection. In our experiments, we use random sampling as reference. It is shown that the customized ATPG-based approaches are the best performing but only offer limited improvement over random sampling. Moreover, their efficacy also diminishes as TroLL's triggers become longer (*i. e.* have more bits specified). We thereby highlight the need to find a scalable HWT detection approach for TroLL.**

***Index Terms*—Logic Locking, Hardware Trojans, ATPG**

I. Introduction

The fact that most chip designers outsource the production of their chips to off-shore foundries raises concerns about the privacy of the chip's intellectual property (IP) and the integrity of the fabrication process. There has been a significant amount of research in both topics. For IP protection, numerous design obfuscation techniques have been proposed to mitigate attacks such as counterfeiting and over production, among which logic locking is by far the most prominent and well-studied class of protection techniques [1]. Logic locking adds key inputs and key-controlled gates into the circuit to make the locked circuit's functionality key-dependent. As the correct key is not known to the untrusted foundry, neither is the correct functionality, and hence the privacy of the design is preserved. Pertaining to the integrity of fabrication, the term Hardware Trojans (HWT) is often used to describe stealthy malicious modifications in the design. Logic locking and HWT's have been studied mostly independently so far. Existing literature have studied the impact of logic locking and HWT's on each other, such as how to use design obfuscation to prevent HWT insertion [2]–[8] and inserting HWT's in logic locked circuits [9], [10]. These works treated logic locking and HWT's as two separate subjects and do not study their similarity. In contrast, in this work, we uncover the structural similarity between logic locking and HWT's. Leveraging this insight, we will discuss how to utilize logic locking techniques to construct novel HWT's. This is a significant threat because logic locking techniques have become public knowledge and adversaries can leverage the principles and structures of logic locking to craft novel HWT's. However, this threat has been largely overlooked in the current security landscape. Our research highlights how adversaries can exploit logic locking mechanisms to create hard-to-detect Trojans. We also explore how to convert attacks against design obfuscation to HWT detection techniques. We demonstrate that such Trojans are resilient not only to existing detection techniques but also to newly customized methods tailored to detect such Trojans. The contribution of this work is as follows.

- We present a novel perspective on logic locking by identifying structural similarities between logic locking schemes and hardware Trojans (HWT's). In particular, we decompose logic locking mechanisms into a *Mutation Unit (MU)* and a *Restore Unit (RU)*, laying the groundwork for cross-domain insights.
- Building on this insight, we propose a new class of Trojans, Trojans based on Logic Locking (TroLL), by embedding only the MU into the circuit and bypassing the RU. This departs from traditional HWT design strategies that rely on rare-event triggers.
- We adapt state-of-the-art Automatic Test Pattern Generation (ATPG) techniques to account for TroLL's trigger characteristics and assess their effectiveness against this new class of Trojans.
- We further extend logic locking attack strategies, specifically SAT-based attacks, to evaluate their applicability in detecting both TroLL and conventional HWTs.
- Our experimental results reveal that TroLL exhibits strong resilience to existing ATPG-based detection techniques such as statistical test generation [11] and maximum

978-1-6654-3274-0/21/$31.00 © 2021 IEEE

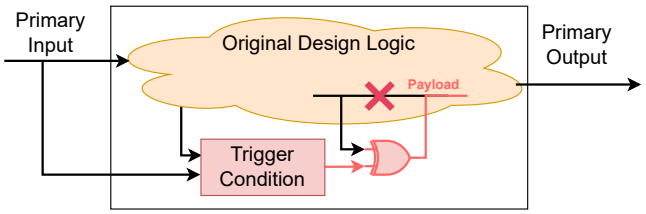


Fig. 1. Illustration of an HWT-infested Circuit

clique sampling [12]. While adapted detection methods perform better on TroLL without sacrificing effectiveness on conventional HWTs, their performance degrades significantly as the trigger length increases.

The rest of this paper is organized as follows. In Section II, we introduce the technical background of hardware Trojans and logic locking. Section III presents the structural similarities between logic locking techniques and hardware Trojans, and the construction of TroLL based on such similarities. The evolved ATPG-based detection approaches and the adaptation of SAT-based attacks on logic locking to HWT detection is formulated in Section IV. In Section V, we present the experiment details on TroLL and the results on detecting TroLL with approaches based on ATPG, SAT, and random sampling. Lastly, we conclude the paper in Section VI.

II. Background and Related Work

In this section, we provide relevant background and survey related efforts in three broad categories. First, we describe the working principle of hardware Trojans. Next, we survey existing test generation efforts for detection of hardware Trojans. Finally, we provide an overview of logic locking techniques.

A. Hardware Trojans

Hardware Trojans (HWT) are stealthy malicious modifications to hardware designs. HWT's usually consist of two components, trigger and payload. The trigger is a condition that activates the HWT, and the payload is the effect of the HWT once activated. The trigger condition can be determined by the circuit's input and/or internal signals in the original design. The HWT payload can have various possible effects, including functionality corruption [13], information leakage [14]–[16], denial-of-service [17], bypass of security properties [18], etc. An illustration of an HWT-infested circuit is given in Fig. 1 where the relationship between the original design and the HWT's trigger and payload is shown.

HWT's can be inserted in almost any phase in the VLSI hardware supply chain, including in third-party IP cores, by a malicious CAD tool, by a rogue designer, by an untrusted fabrication facility, etc. [19], [20]. The HWT's inserted before the fabrication stage are present in the design files (*e.g.* RTL and/or gate-level netlists). Therefore, it is possible to use formal methods, such as logic equivalence checking, to tell whether an HWT exists [21], [22]. However, for HWT's inserted by the foundry, the netlist of the HWT-infested circuit is not available to the designer. Some researchers have proposed to use reverse engineering to obtain the layout of the HWT-suspicious chip [23], [24]. However, IC reverse engineering is increasingly expensive and error-prone as technology node scales down [25] To date, the smallest technology node that is publicly known to be successfully reverse engineered down to the netlist at the whole chip scale is 14nm [26], and the smallest node demonstrated for reverse engineering-based HWT detection is 20nm [27]. The netlist in state-of-the-art (SOTA) technology node is at 3nm [28], [29], which means the reverse engineering-based HWT detection toolchain is a few generations behind, making it infeasible to perform netlist-based HWT detection for chips produced at the latest nodes. Hence, testing is the most practical way to detect HWT's inserted by untrusted foundries. Besides, testing-based methoes are also applicable to HWT's inserted by IP providers, CAD tools, rogue designers, etc. The state-of-the-art automatic test pattern generation (ATPG) approaches for HWT detection will be introduced in Section II-B.

B. ATPG-based HWT Detection

Both combinational and sequential HWT triggering mechanisms have been proposed in the literature. However, since the designer likely has access to testing infrastructure that allows the circuit to be tested combinationally (*e.g.* scan-chain), a sequential HWT trigger can be broken down into a series of combinational ones. We hence focus on combinational HWT triggers in this work. State-of-the-art combinational HWT insertion methodology utilizes rare signals (*i.e.* an internal node's value that is functionally difficult to sensitize) as the trigger, ensuring that the HWT is only triggered in rare circumstances [30], [31]. Based on this property, many HWT detection methods have been developed based on ATPG principles. Existing approaches explored two complementary directions when dealing with test generation for activation of rare signals: 1) statistical test generation, and 2) directed test generation. A promising avenue for statistical test generation is to rely on N-detect principle [32] by activating each rare signal N times to increase the statistical likelihood of activating the unknown trigger in the HWT. MERO [11] tries to generate test vectors to activate the same rare signal N times by flipping input vector bits one at a time. Saha *et al.* improved the test generation performance using genetic algorithm and Boolean satisfiability [33]. Pan *et al.* improved the performance further by flipping bits using reinforcement learning [34].

While N-detect methods try to activate one rare signal at a time, Lyu *et al.* focused on activating as many rare nodes as possible using maximal clique sampling (TARMAC [12]). TARMAC first creates the satisfiability graph for all the rare signals. In this graph, each vertex stands for a rare signal, and there is an edge connecting two vertices if and only if there exists an input pattern that sensitizes the two rare signals simultaneously. Next, the maximal cliques from the satisfiability graph is computed. Finally, TARMAC generates tests to activate randomly sampled set of maximal cliques. If any of the generated tests is able to activate the trigger, the HWT will be detected.

C. Logic Locking

Logic locking has emerged as a protection mechanism against potential piracy and overbuilding threats in untrusted

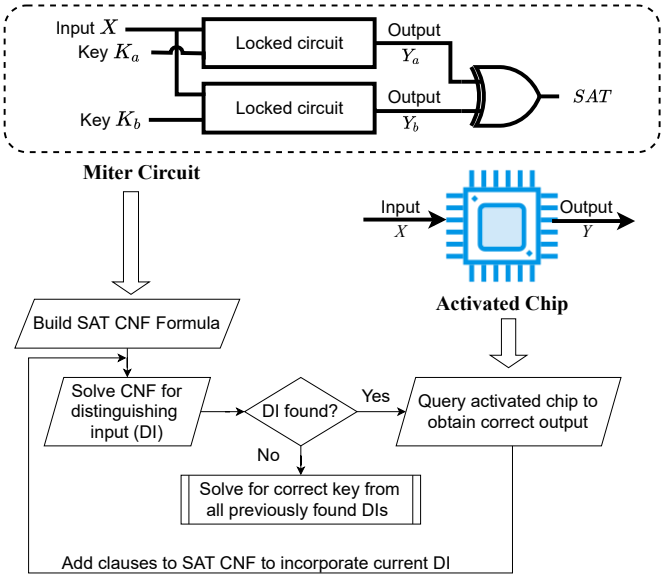


Fig. 2. The basic procedure of SAT-based attacks

fabrication facilities. These techniques obfuscates the hardware by adding key inputs into the circuit without disclosing the correct key to the fab. Hence, the fab will not know the full functionality of the design. When the fabrication is done, the chip designer (or a trusted third party) will provide the key to the chip by connecting a piece of tamper proof memory. This process is called *activation*. This way, only the authorized users will have access to an *activated chip* which has the correct functionality.

There have been many attacks formulated against logic locking, among which the ones based on Boolean satisfiability theory, a.k.a. SAT-based attacks [35], have both mathematical guarantee to find the correct key and strong practical performance. The flow of SAT-based attacks is demonstrated in Fig. 2. As demonstrated, a miter circuit is built. The miter contains two copies of the locked netlist that share the same input but are keyed separately. Their outputs are XOR'ed. Essentially, if the miter's output is $TRUE$, the input is causing different outputs with the two keys. The SAT-based attacks are iterative. In each iteration, a Boolean satisfiability problem is solved to find an input pattern and two keys that satisfy the miter circuit. The input pattern is called the *distinguishing input (DI)*. The activated chip is then queried to get the correct output value. Then, clauses are added to the miter-based SAT formula so that all the wrong keys that causes an incorrect output for the DI are pruned out. A correct key will be found when the DIs found have pruned out all the wrong keys.

Many SAT resilient logic locking techniques have been proposed to thwart the attack. In this work, we will examine these techniques and summarize the structural similarities among them. We then show how these logic locking techniques can guide the construction of novel hardware Trojans.

III. Locking Inspired HWT Construction

In this section, we provide a brief overview of existing obfuscation art. We then explore how the properties of these techniques can be leveraged in order to construct difficult-to-detect HWT's by slightly modifying their logical topologies, maintaining their rigorous mathematical guarantees but retargeting them to HWT application. The intuition behind such conversion is that, for both locking and Trojan, error is injected into a circuit only when the circuit has a specific input pattern:

- For locking: The input pattern is among those that are corrupted by the given wrong key.
- For Trojans: The input pattern matches the trigger.

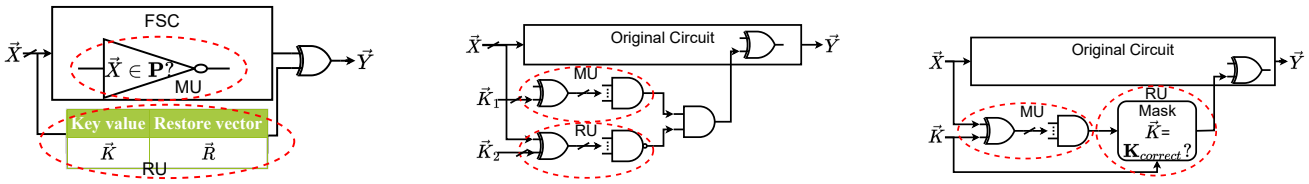
Because HWT's should be triggered only by very few input patterns to evade detection [11], [12], the logic locking schemes suitable for converting to HWT's should also corrupt very few input patterns given a wrong key. Such logic locking techniques do exist and they are mainly designed to thwart SAT-based attacks. These techniques include Anti-SAT [36], SARLock [37], stripped functionality logic locking (SFLL) [38], Robust Strong Anti-SAT [39], CASLock [40], etc. In this work, we first analyze the commonality among these locking approaches. Next, we present the HWT construction based on these locking algorithms.

A. Commonality among Logic Locking

No matter how distinct these logic locking constructions seem to be, we find that they can all be decomposed into two functional units that interact together to inject error for specific input pattern given a wrong key. We call them a *Mutation Unit (MU)* and a *Restore Unit (RU)*. Essentially, the MU modifies the circuit's functionality for some input patterns and the RU tries to restore the modified functionality. When the correct key is applied, the RU restores the correct input patterns modified by the MU and so the locked circuit produces correct output to all input values. When the key is incorrect, however, the error injected by the MUs will not be corrected by the RU. In this case, if the input's functionality is modified by the MU, its output will be corrupted. The number of input patterns modified by the MU should be very small in order for the logic locking approach to be resistant to SAT-based attacks [41]. The rarity of such input patterns makes them suitable for HWT triggers. We use SFLL, SARLock, and Anti-SAT as examples of SAT-resilient locking techniques and briefly review how the MU and RU interact in each of them.

1) Stripped Functionality Logic Locking (SFLL): Fig. 3a shows the block diagram of SFLL. It is composed of two parts: a functionality stripped circuit (FSC) and a restore unit (RU). The FSC is the original circuit with the functionality altered for a set of protected input patterns (PIP), denoted as $\mathbf{P}$. The FSC's internal structure that modifies the functionality of the PIPs is the MU of SFLL. Notice that RU of SFLL coincides with our general definition of RU. The structure of the RU in SFLL is a look-up table (LUT). If the circuit's input matches the LUT key, the LUT will produce a restore signal. If the LUT contains the correct key, the restore signal will reverse the corruption caused by the FSC. If the LUT contains an incorrect key, both the PIPs and the input patterns that correspond to the key will be corrupted.

2) Anti-SAT: The structure of Anti-SAT is shown in Fig. 3b. The MU and the RU have similar structure. For the MU, there is an array of XOR gates followed by an AND tree. Depending on $\vec{K}_1$, there is only one input value of $\vec{X}$ such



(a) SFLL (b) Anti-SAT (c) SARLock

Fig. 3. MU and RU in Logic Locking Constructions

that the MU will evaluate to logic 1. Let us call this value $\vec{X}_M$. The RU's structure is very similar to the MU's, and the only difference is that the AND tree's output is inverted. Depending on $\vec{K}_2$, there is only one input value of $\vec{X}$ that will make the RU evaluate to logic 0. Let us call this value $\vec{X}_R$. Corruption is injected into the circuit when both the MU and the RU evaluate to logic 1, *i.e.* when $\vec{X} = \vec{X}_M$ and $\vec{X} \neq \vec{X}_R$. Therefore, a correct key must be such that $\vec{X}_M = \vec{X}_R$. This way, the RU will output logic 0 when MU outputs logic 1 and prevent the original circuit from being corrupted.

3) SARLock: SARLock also contains an MU and an RU, as shown in Fig. 3c. Its MU is the same as the one in Anti-SAT: depending on the key, there is one input value that will let the MU evaluate to logic 1. The RU checks if the key input contains the correct key. If so, it will mask the MU's output and prevent it from corrupting the original circuit.

B. Advantages of Locking based Trojans

In each of the above-mentioned logic locking techniques, the MU is capable of injecting error into the circuit, and the RU will prevent the error from being injected if the correct locking key is provided. We notice that the MU naturally offers properties desirable for the trigger of HWT's:

- Corruption is injected for very few (or just one) input pattern in the exponential input space, which makes random sampling based detection very difficult.
- The corrupted input patterns need not have any correlation with the original netlist's structure, so that they can be chosen to avoid ATPG or rare signal based detection approaches such as [11] and [12].
- These trigger patterns are completely known to the attacker. Contrarily, enumerating triggers of conventional rare signal based Trojans is mathematically intractable in general because it is a satisfiability counting problem [42]. Hence, it is much easier for the attacker to control when to trigger the Trojan and avoid unintended triggering using TroLL.

C. Construction of TroLL

These properties indicate that the MU's of logic locking can serve as ideal HWT trigger circuitry. Building upon this discovery, we present Trojans based on logic locking (TroLL), which employs the MU of logic locking to modify the functionality of the original circuit. We present a generalizable way to convert a logic locking technique to TroLL as follows:

1) Identify MU and RU in the locked netlist and remove the RU. Hard-code the RU's output value to the one that does not invert the output of the MU.
2) If the MU has a key input (such as Anti-SAT), hard-code the key such that the desired HWT trigger can cause the MU to corrupt the circuit.

Essentially, when building TroLL from SFLL, we only need to remove the RU and make sure that the PIP's represent the Trojan trigger patterns we want. For Anti-SAT and SARLock, we need to remove the RU and hard-code the MU keys to incorporate the triggers. E.g., for the Anti-SAT construction in Fig. 3b, we need to remove the RU and fix its output at logic 1. For the MU, we fix $\vec{K}_1$ to be the bitwise-inverted trigger pattern. A constant sweep is then performed to simplify the circuit. In this way, the key inputs of logic locking will be all removed and the TroLL-infested circuit has the same I/O pins as the original circuit. No matter which logic locking technique TroLL is made from, the functionality of TroLL will be identical. Besides, as each of the above steps is a strict subtraction from logic locking infrastructure, TroLL's overhead will be much lower than that of logic locking. Notice that, although we describe a gate-level operation to build TroLL in the above example, TroLL can be incorporated at RT or behavioral level using the two-step process as well.

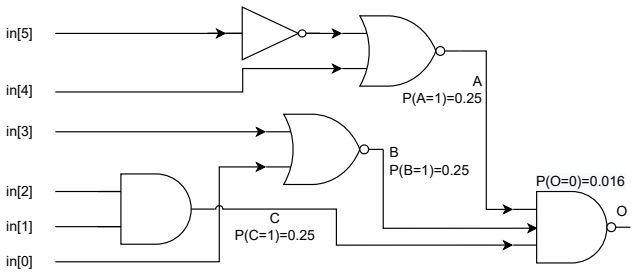
D. Choosing TroLL Trigger Pattern

TroLL needs to evade HWT detection. As introduced in Section II-A, existing state-of-the-art HWT detection approaches find test patterns that sensitize rare signals in the original design. To evade these detection approaches, TroLL trigger patterns need to avoid sensitizing any rare nodes. To begin with, we use a random sampling approach to determine the rare value of each internal node, r_i, and its associated probability, p_i. Although an alternative to random sampling is the signal probability skew analysis [43], the complexity of such analysis often increases exponentially if the correlation between signals is to be accounted for [44]. Then we use Algorithm 1 to determine the trigger pattern for TroLL. Essentially, the algorithm finds an input pattern with the maximum probability threshold p_{max} such that no rare value below this probability will be realized by the trigger. Such a process is illustrated in Fig. 4. In the sample circuit, the rare values and their probabilities are annotated for each internal node. A list of randomly generated input patterns are shown under the circuit diagram. The signal sensitized by each input pattern that has the lowest probability are highlighted in pink. Algorithm 1 will choose the input pattern that maximizes the lowest probability. In this example, the trigger pattern will be the one in the last row since it does not sensitize any rare value. TroLL triggers selected by this process will be immune

Algorithm 1: TroLL Trigger Selection

input : $G_1 \dots G_n, r_1 \dots r_n, p_1 \dots p_n$; // *The Boolean function, rare output value and associated probability of each gate in the original design*
input : $\mathcal{S}$; // *A set of random input vectors*
output: X_T ; // *The best TroLL trigger found*
output: p_{max} ; // *Maximum rare value probability threshold*

Initialization: $p_{max} \leftarrow 0$;
foreach $X \in \mathcal{S}$ **do**
  $p_{tmp} \leftarrow 0.5$; // *tracking minimum rare probability for each sample*
  foreach $i \in \{1 \dots n\}$ **do**
    // *Check if gate i has the rare value*
    if $G_i(X) == r_i$ **then**
      if $p_{tmp} > p_i$ **then**
        $p_{tmp} \leftarrow p_i$;
      end
    end
  end
  if $p_{tmp} > p_{max}$ **then**
    $p_{max} \leftarrow p_{tmp}$;
    $X_T \leftarrow X$;
  end
end



Example: Choosing TroLL Trigger from Randomly Generated Input Patterns using Algorithm I

in[5]	in[4]	in[3]	in[2]	in[1]	in[0]	A	B	C	O	p_{max}
1	0	0	1	1	0	1	1	1	0	0.016
1	0	1	0	1	0	1	0	0	1	0.25
1	1	0	1	0	0	0	1	0	1	0.25
0	0	0	1	1	1	0	0	1	1	0.25
0	1	1	1	0	0	0	0	0	1	0.5

Fig. 4. Illustration of how to Choose TroLL Trigger using Algorithm 1 on a Sample Circuit

to the existing rare value based detection approaches such as those introduced in Section II-A.

The fact that TroLL triggers does not sensitize any rare signal does not mean that TroLL can be triggered by high probability signals or can be easily detect by random sampling. On the contrary, TroLL is essentially creating a new rare node that only the trigger pattern can sensitize. Since the defender does not have the netlist of the HWT-infested circuit and can only base the detection on the original circuit, they do not have any information about the new node and hence cannot generate test patterns aimed at sensitizing the new node. Also notice that the triggers selected using Algorithm 1 has the full input length. This will likely cause high overheads. As we later demonstrate in Sections V-C, practical resilience against HWT detection can be attained when only a subset of input bits are taken as the TroLL trigger.

IV. Detection Techniques for TroLL

In this section, we introduce a few novel approaches that are aimed to detect TroLL more effectively. The first type of approaches are based on the trigger selection process of TroLL: by avoiding any test pattern that sensitize any rare node value, ATPG-based HWT detection mechanisms will generate test patterns that are more likely to match the trigger of TroLL. The second approach is based on the fact that TroLL originates from logic locking, and SAT-based attacks are the most formidable attacks on logic locking. Therefore, we can formulate a Trojan detection approach that emulates the a SAT-based attack on logic locking.

A. Customizing ATPG-based HWT Detection Approaches for TroLL

Given TroLL's trigger selection mechanism, we can customize existing ATPG-based HWT detection approaches to detect TroLL. TroLL's trigger selection process eliminates any input pattern that sensitizes any rare internal node value as described in Algorithm 1. The same principles can be applied to the test generation algorithms for HWT detection: instead of targeting the rare values, the ATPG algorithms can choose the test patterns that satisfy as many prevalent values as possible. Following the notations used in Algorithm 1: say that n internal nodes of a combinational circuit that implement Boolean functions $G_1 \dots G_n$ have rare values $r_1 \dots r_n$ that are below a certain threshold p where $0 < p < 0.5$. In other words, these n nodes have prevalent values $\bar{r_1} \dots \bar{r_n}$ that have probabilities above $1 - p$. While existing HWT detection algorithms aim to find test patterns X that satisfy as many $G_i(X) = r_i$ as possible ($i = 1 \dots n$), a TroLL-specific detection algorithm should instead find input patterns that satisfy $G_i(X) = \bar{r_i}$ for as many i as possible.

Given such a principle, it is surprisingly convenient to customize existing HWT detection approaches for TroLL. We can indeed run the same ATPG algorithms, such as statistical test generation [11] or maximal clique sampling [12], and target the same set of internal nodes. The only change is to invert the targeted Boolean values of these nodes. Statistical test generation (such as N-detect) can target to generate test vectors to activate each prevalent node value N times, whereas maximal clique sampling can build the satisfiability graph on the prevalent values instead of the rare values.

Because the defender does not know the type of HWT when the test patterns are generated, the test patterns should be able to detect conventional HWTs as well. Therefore, for each ATPG algorithm, we combine the test patterns that are generated to sensitize the rare values (for conventional Trojans) and those generated to avoid sensitizing rare values (for TroLL). We refer to such an approach as *Evolved Statistical Test Generation* and *Evolved Maximal Clique Sampling*. In Section V-C, we will present the efficacy of these evolved HWT detection approaches.

B. Adapting SAT-based Attacks on Logic Locking for HWT Detection

Attacks on logic locking try to find the correct key, whereas Trojan detection aims to find the trigger of HWT's. Since TroLL is based on logic locking, it is natural to associate logic locking attack with the detection of TroLL. However, since the defender does not know which type of HWT is potentially inserted, the detection approaches must not be limited to TroLL but generalizable to any type of HWT. In this section, we present how to adapt the SAT-based attacks on logic locking to detecting HWT's. A SFLL-like auxiliary circuit will be constructed based on the HWT-suspicious circuit where the Trojan's trigger and payload are represented by keys. Then, the SAT attack formulation is used to find a key that can represent the HWT. The HWT is detected when such a key is found. In Section V, this SAT-based detection approach as well as the ATPG-based approaches will be used to evaluate the detectability of TroLL and conventional HWT's.

1) Construction of the Auxiliary Circuit: A defender has the netlist of the original circuit and the fabricated HWT-suspicious circuit. The netlist of the fabricated circuit is not available. In order to search for a trigger pattern, an SFLL-like auxiliary circuit to emulate an HWT-infested circuit is constructed. As shown in Fig. 5, the auxiliary circuit is built by adding a look-up table to emulate the trigger and payload of the HWT. The trigger key K_T is compared with the circuit input X. When they are the same, the payload key K_P is bit-wise XOR'ed with the output Y.

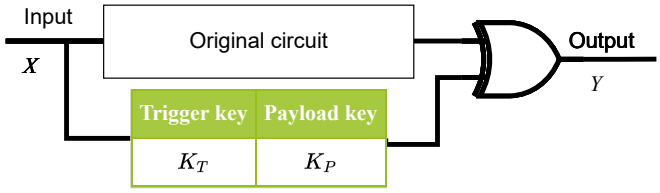


Fig. 5. Construction of the auxiliary circuit for SAT-based detection

Note that SAT-based detection does not assume any knowledge information about the potentially existing HWT, and the construction of the auxiliary circuit is independent from the actual trigger and payload of the HWT. The purpose of the auxiliary circuit is to emulate the trigger and payload of HWT's rather than being functionally equivalent to the HWT-suspicious circuit. Since only one trigger needs to be found to detect the HWT, we only need to have one entry in the LUT of the auxiliary circuit.

2) Detection Flow: The flow of SAT-based Detection is laid out in Fig. 6. Similar to the SAT-based attack against logic locking introduced in Section II-C, a miter circuit is built using two copies of the auxiliary circuit and their outputs are XOR'ed. Let $F(\vec{X})$ be Boolean function of the original circuit, $F_A(\vec{X}, \vec{K}_T, \vec{K}_P)$ be that of the auxiliary circuit, and $H(\vec{X})$ be that of the HWT-suspicious circuit. In the first iteration, the following SAT formula is solved to obtain the distinguishing input (DI):

$$F_A(\vec{DI}_1, \vec{K}_{Ta}, \vec{K}_P) \neq F_A(\vec{DI}_1, \vec{K}_{Tb}, \vec{K}_P) \quad (1)$$

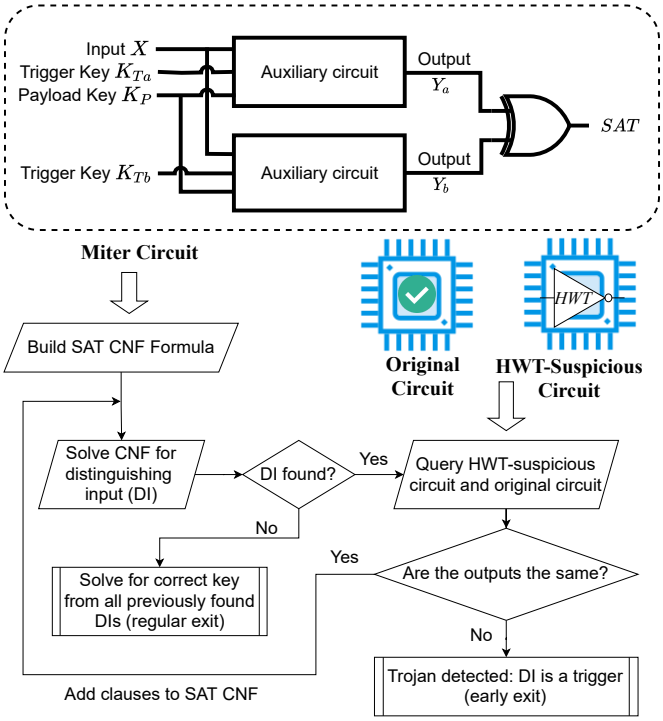


Fig. 6. The SAT-based HWT Detection Flow

The subscript of DI stands for the iteration number. Then, both the original circuit and the HWT-suspicious circuit are queried with the DI. If the results are not equal, *i.e.* $F(DI_1) \neq H(DI_1)$, then the HWT is detected and DI_1 is an HWT trigger. If they are equal, then let $O_1 = H(DI_1)$. In the second iteration, clauses are added to ensure that the new keys found should produce correct output for DI_1 since it is not the trigger:

$$\begin{aligned} &F_A(\vec{DI}_2, \vec{K}_{Ta}, \vec{K}_P) \neq F_A(\vec{DI}_2, \vec{K}_{Tb}, \vec{K}_P) \\ \bigwedge &F_A(\vec{DI}_1, \vec{K}_{Ta}, \vec{K}_P) = F_A(\vec{DI}_1, \vec{K}_{Tb}, \vec{K}_P) = O_1 \end{aligned} \quad (2)$$

The added clause will exclude *any* trigger key K_T that mistakes a non-trigger $\vec{DI}_1$ as a trigger, which makes SAT-based detection potentially more efficient than purely testing-based detection approaches which only determine whether the test pattern is an HWT trigger or not.

The process of SAT-based detection have some key differences from the SAT attack on logic locking:

- The oracle used in the formulation is the HWT-suspicious circuit under detection, instead of an activated chip.
- An early exit condition is added. If the DI produce a different output on the HWT-suspicious circuit compared to the original circuit, the detection process will terminate because an HWT is detected.
- The same payload key is applied to both copies of the auxiliary circuits to ensure that the output difference of the two copies is caused by the trigger key.
- The correct key found by SAT attack on logic locking will make the locked circuit have the same functionality as the original circuit, whereas the SAT-based detection is not meant for replicating the exact functionality of the HWT-free circuit.

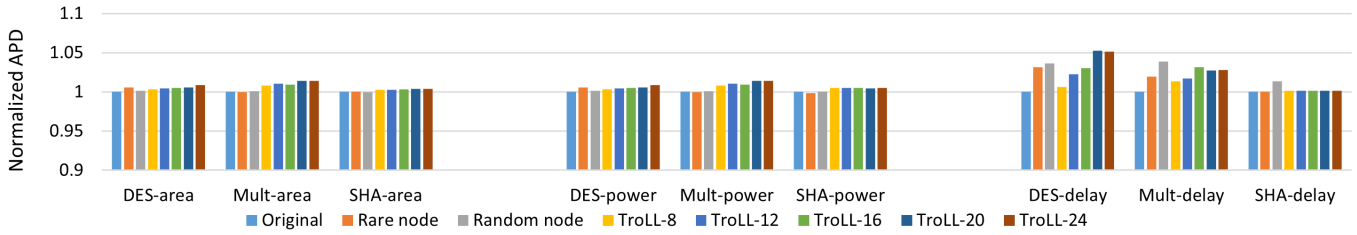


Fig. 7. Mean area, power, and delay overhead of each HWT type

C. Applicability of Netlist-based Attacks on Logic Locking for TroLL Detection

There are attacks on logic locking that analyze the locked netlist for structural and functional information and achieve better results than the SAT attack. For example, Sirone et. al. proposed netlist analysis methodology to identify the MU's in SFLL and extracted the correct keys [45]. The work of Han et. al. automated such analysis using Electronic Design Automation (EDA) tools and was applicable more generally to most logic locking methods based on the corrupt-and-correct principle [46]. These attacks are predicated on the attacker having access to the locked netlist of the circuit. From the netlist, the attacker can exploit structural and functional properties of the underlying logic locking approach to infer the correct key or to identify vulnerabilities.

In contrast, our threat model assumes that the defender (e.g., the design house) does not have access to the detailed netlist of the fabricated chip, especially if the fabrication process is outsourced to an untrusted foundry. Although our TroLL design borrows techniques from logic locking, the "roles" played the attacker (foundry) and defender (designer) are reversed. In logic locking, the designer inserts key gates and provides a locked netlist to an untrusted foundry. The foundry knows the modified netlist and tries to uncover the key. By contrast, in foundry-inserted hardware Trojans (HWT), the foundry secretly modifies the designer's netlist, and the designer generally does not know the final netlist. If the designer wants to obtain the netlist of designs with Trojans inserted during fabrication, reverse-engineering the finished chip is required. This is a costly, destructive, and slow process [25]. On the other hand, if reverse engineering were performed, simple netlist comparison would reveal any deviation, making the attack unattractive to an adversary intent on stealth. Therefore, the mainstream HWT literature does not consider the defender to have access to the netlist [11], [12], [32]–[34]. We summarize the threat models considered in logic locking and HWTs side-by-side in Table I for a clearer comparison.

In summary, although TroLL leverages logic-locking structures, its attacker/defender assumptions align with the established HWT threat model, not with logic locking's. This lack of access precludes the application of netlist-based attacks like [45], [46] for Trojan detection. Instead, defenders typically rely on test-based approaches to detect anomalies in circuit behavior.

TABLE I
CONTRASTING THREAT MODELS FOR LOGIC LOCKING AND FOUNDRY-INSERTED HARDWARE TROJANS.

	Logic Locking	**HWTs**
Netlist modified by	Designer	Foundry
Who tries to decipher the change?	Foundry	Designer
Designer knows final netlist?	Yes	No
Foundry knows original netlist?	No	Yes

TABLE II
BENCHMARKS USED IN HWT EVALUATIONS

Benchmark Name	# Gates	# Inputs	# Outputs
DES	6,473	256	245
32-Bit Multiplier	10,609	64	64
SHA-256	51,222	512	256

D. Summary

In this section, we introduce two types of novel HWT detection techniques that have potentials to detect TroLL more effectively than existing approaches. The evolved ATPG-based detection aims at finding the trigger based on TroLL's trigger selection algorithm, whereas SAT-based detection is an effort to take advantage of TroLL's resemblance to logic locking. We also clarify that netlist-based attacks on logic locking, albeit being more effective attacks, cannot be adapted to detect locking-based Trojans due to the lack of netlist access. In the next section, we will examine the customized detection techniques alongside the existing ones to evaluate TroLL's ability to evade detection.

V. EXPERIMENTS

In this section, we present details on TroLL implementation and evaluation. We also compare the detection approaches introduced in Section IV with existing state-of-the-art ATPG-based HWT detection approaches and random sampling on both TroLL and conventional HWT's.

A. Experiment Setup

In our experiments, we implement both TroLL and conventional hardware Trojans, including rare node triggered Trojans and random node triggered Trojans. Three circuit benchmarks are used for the evaluation: DES, a 32-bit multiplier, and SHA-256, with a range of sizes as shown in Table II. Each benchmark is in gate-level Verilog format and all HWT's are

incorporated by modifying the Verilog file. We use Icarus Verilog as the simulation tool. In Section V-B, we describe which Trojans instances are made and their cost in terms of area, power and delay.

For HWT detection, we emulate post-fabrication testing by simulating the HWT-infested circuit and comparing the observable nodes, including output pins and flip-flop values, with the correct values. The simulation tool is also Icarus Verilog. This is consistent with real chip testing where the output pins can be read directly, the flip-flop values can be read through the scan chain, and other internal nodes inside combinational logic cones cannot be read directly. In Section V-C, we present HWT detection results using multiple test-based methods.

B. Trojan Implementation and Overhead

We analyze and determine the rare values and associated probability of each internal node by simulating 100,000 randomly generated input patterns on each benchmark circuit. For rare node triggered Trojans, the triggers are selected directly based on this analysis. For TroLL, we choose trigger patterns using Algorithm 1 introduced in Section III-D. Notice that the length of these triggers are the same as the circuit's input. When a shorter trigger length is needed, we choose a random subset of bits from the trigger patterns. For the HWT payload, we choose a subset of output pins to flip when the trigger condition is satisfied and the payload is the same across all the HWT instances for the same benchmark. Incorporating the MU output into the primary outputs guarantees that any corruption introduced is directly reflected in the observable behavior, minimizing the risk of the malicious effect being masked or rendered latent during internal signal propagation, which is desirable for the attacker. To improve the stealthiness of TroLL, the attacker can increase the trigger length, which will significantly decrease the likelihood of the trigger being present in the test patterns for Trojan detection.

TroLL with 8, 12, 16, 20, and 24 bits trigger are implemented. For each type of HWT, we create 100 instances. The area, power and delay (APD) values are evaluated through synthesis in a 45nm library using Synopsys Design Compiler. The average APD values of each benchmark is shown in Figure 7. The APD values are normalized using those of the original design of each benchmark. For example, a value of 1.01 indicate a 1% overhead. As we can observe from the figures, the overhead percentage of TroLL is very low, especially for the largest benchmark (SHA-256). Hence, TroLL can be very well hidden in large designs and it is very difficult to identify them by APD analysis. This also underscores the scalability of TroLL.

C. HWT Detection

In this section, we evaluate the effectiveness of HWT detection techniques: ATPG-based methods, including statistical test generation (MERO [11]) and maximal clique sampling (TARMAC [12]), the evolved versions of statistical test generation and maximal clique sampling, SAT-based detection, and random sampling. Figure 8 provides an overall comparison

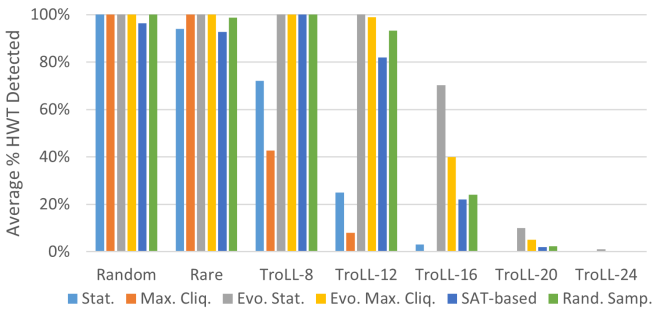


Fig. 8. Comparison of HWT Detection Approaches

of the efficacy of these detection approaches. In the figure, the percentage of detected HWTs is clustered by the HWT type, including rare node Trojans, random node Trojans, and TroLL with 8 to 24 bit triggers. For a certain HWT type and size, the percentage of HWTs detected is calculated by dividing the number of HWTs detected by the total number of HWTs of the same type and size. In general, the detected percentage of any approach drops significantly as TroLL's trigger length increases. This underscores TroLL's resilience to test-based HWT detection approaches. We can also see the performance difference among the detection approaches. Overall, the conventional ATPG-based HWT detection approaches have the worst efficacy against TroLL. However, their evolved versions, especially evolved statistical sampling, outperform all other methods on TroLL with 12 to 20 bits in the trigger. The detection percentage of random sampling and SAT-based detection lie between the original and evolved ATPG-based detection approaches.

Table III shows the percentage of HWT's detected using six test-based methods. The results of the original ATPG-based methods, including statistical methods [11] and maximal clique sampling [12] are shown on the top section of Table III. For rare and random node triggered Trojans, these methods can detect 100% of the HWT's in most cases. For TroLL, these detection methods are not effective as the detection percentage fades quickly with the increase in trigger length. This is because TroLL's trigger selection algorithm, as presented in Section III-D, intentionally avoids sensitizing any rare nodes within the original circuit. As both statistical test generation [11] and maximal clique sampling [12] ensures that each test pattern will sensitize some rare nodes in the circuit, they are unlikely to sensitize the triggers of TroLL.

In the middle section of Table III, we show the HWT detection results with the evolved ATPG-based approaches. Compared to the original ATPG-based approaches, the evolved ones are able to detect more TroLL-type HWTs. Specifically, the improvement in detection percentage is most significant for TroLL with trigger length up to 20 bits. In these detection methods, thanks to the knowledge to TroLL's trigger selection mechanism, the defender can set the targeted node values to meet TroLL's trigger criteria when using ATPG algorithms to generate test patterns. This is the main cause of the improvement.

SAT-based detection is implemented based on the code framework of SAT-based attacks on logic locking presented

TABLE III
HARDWARE TROJAN DETECTION PERFORMANCE BY EXISTING AND PROPOSED METHODS

Benchmark	Statistical Test Generation [11]							Maximal Clique Sampling [12]						
	Random	Rare	TroLL with trigger length					Random	Rare	TroLL with trigger length				
			8	12	16	20	24			8	12	16	20	24
DES	100%	100%	38%	5%	0%	0%	0%	100%	100%	46%	8%	0%	0%	0%
Mult-32	100%	83%	100%	62%	8%	1%	0%	100%	100%	55%	15%	1%	0%	0%
SHA-256	100%	100%	77%	9%	1%	0%	0%	100%	100%	27%	0%	0%	0%	0%

Benchmark	Evolved Statistical Test Generation							Evolved Maximal Clique Sampling						
	Random	Rare	TroLL with trigger length					Random	Rare	TroLL with trigger length				
			8	12	16	20	24			8	12	16	20	24
DES	100%	100%	100%	100%	59%	11%	1%	100%	100%	100%	99%	45%	6%	0%
Mult-32	100%	99%	100%	100%	85%	10%	0%	100%	100%	100%	99%	41%	4%	0%
SHA-256	100%	100%	100%	100%	67%	8%	1%	100%	100%	100%	100%	34%	4%	0%

Benchmark	SAT-based Detection							Random Sampling						
	Random	Rare	TroLL with trigger length					Random	Rare	TroLL with trigger length				
			8	12	16	20	24			8	12	16	20	24
DES	94%	84%	100%	100%	26%	1%	0%	100%	100%	100%	99%	31%	1%	0%
Mult-32	100%	96%	100%	82%	23%	4%	0%	100%	96%	100%	95%	10%	1%	0%
SHA-256	95%	98%	100%	70%	16%	1%	1%	100%	100%	100%	86%	33%	5%	0%

in [35]. We limit the time of each SAT-based detection run to 48 hours and a Trojan is considered as not detected if no trigger pattern is found within this time frame. In the bottom left division of Table III, we show the percentage of HWT detected by SAT for each benchmark and type of HWT. The efficacy of this approach is better than statistical methods [11] and maximal clique sampling [12] but not as good as their evolved counterpart. A possible reason is that the TroLL avoids using rare nodes as triggers. As these rare nodes values are inherently hard to satisfy, not using them improves the efficacy of SAT-based detection.

The random sampling detection results are shown in the bottom right division of Table III. This method can be used as the baseline case as it does not require any specialized algorithm. In Figure 8, it can be seen that the ATPG-based approaches evolved have higher efficacy than random sampling in TroLL, while their original versions perform worse than random sampling. This indicates that the customization of the ATPG-based approaches presented in Section IV-A is effective against TroLL.

D. Discussions

Through the results presented in Section V-C, we can see that, for each existing and proposed ATPG-based detection technique studied, there is a range of trigger bit numbers where the detection percentage drops from 100% to 0. This trend reflects a fundamental limitation of test-based HWT detection approaches. Test-based approaches can detect an HWT only when the test input triggers the HWT and leads to an incorrect output. The input space is exponential in the number of input bits, while only a limited number of test patterns can be applied during the test time frame. Therefore, the test patterns account for a very small subset of the entire input space, forcing the ATPG algorithms to choose test patterns that will most likely trigger the HWT's. Such a likelihood can only be assumed from the HWT insertion algorithms, and any incorrect assumption will diminish the likelihood of successful HWT detection. This can explain the poor performance of using test patterns generated for rare node-triggered HWTs for TroLL. When the correct assumptions are used, as is the case for the evolved ATPG-based approaches for TroLL, the detection percentage becomes better. However, the detection is still test-based, and the test fraction of the input space still becomes exponentially smaller as the number of input bits increases. Therefore, it is unlikely that any purely test-based HWT detection approach can maintain its efficacy with the increase in input bits.

VI. CONCLUSION

In this paper, we present a novel type of Hardware Trojans based on logic locking, TroLL. TroLL is constructed by retaining the mutation unit (MU) and removing the restore unit (RU) of state-of-the-art logic locking techniques. The trigger patterns of TroLL are selected in a way that avoids sensitizing the internal rare signals of the original circuit, thereby evading state-of-the-art ATPG-based detection schemes. In an attempt to formulate an effective detection approach against TroLL, we tried several different approaches, including evolving the ATPG-based approaches targeting the internal nodes' prevalent values in addition to the rare values, and adapting the SAT-based attacks on logic locking to HWT detection. We also use random sampling as a reference. We found that the evolved ATPG-based approaches performed better than random sampling, but even these approaches' efficacy diminishes as TroLL's triggers get longer. Therefore, we have identified TroLL as a new threat to the integrity of hardware manufactured in untrusted fabrication facilities, and it is necessary to find a scalable detection approach against TroLL.

On a broader scale, this paper reminds us that even a design protection scheme (such as logic locking) can be a double edged sword. Meanwhile, just like the SAT attack can be turned to an HWT detection scheme, we can examine other attacks against logic locking in the search for a more effective detection approach against TroLL.

REFERENCES

[1] A. Chakraborty *et al.*, "Keynote: A disquisition on logic locking," *IEEE Transactions on CAD*, vol. 39, no. 10, pp. 1952–1972, 2019.
[2] J. Frey and Q. Yu, "Exploiting state obfuscation to detect hardware trojans in noc network interfaces," in *MWSCAS*, 2015, pp. 1–4.
[3] W. Hu *et al.*, "Leveraging unspecified functionality in obfuscated hardware for trojan and fault attacks," in *AsianHOST*, 2019, pp. 1–6.

[4] Q. Yu *et al.*, "Exploiting hardware obfuscation methods to prevent and detect hardware trojans," in *MWSCAS*, 2017, pp. 819–822.

[5] S. Dupuis *et al.*, "A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans," in *IOLTS*, 2014, pp. 49–54.

[6] Z. Mirmohammadi and S. Etemadi Borujeni, "A new optimal method for the secure design of combinational circuits against hardware trojans using interference logic locking," *Electronics*, vol. 12, no. 5, p. 1107, 2023.

[7] J. Cruz, P. Gaikwad, and S. Bhunia, "Analysis of hardware trojan resilience enabled through logic locking," in *2022 Asian hardware oriented security and trust symposium (AsianHOST)*. IEEE, 2022, pp. 1–6.

[8] F. Wang, Q. Wang, L. Alrahis, B. Fu, S. Jiang, X. Zhang, O. Sinanoglu, T.-Y. Ho, E. F. Young, and J. Knechtel, "Trolloc: Logic locking and layout hardening for ic security closure against hardware trojans," *arXiv preprint arXiv:2405.05590*, 2024.

[9] R. Chakraborty and S. Bhunia, "Security against hardware trojan through a novel application of design obfuscation," in *ICCAD*, 2009, pp. 113–116.

[10] J. Maynard and A. Rezaei, "Reconfigurable run-time hardware trojan mitigation for logic-locked circuits," in *2024 IEEE 17th Dallas Circuits and Systems Conference (DCAS)*. IEEE, 2024, pp. 1–6.

[11] R. S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, and S. Bhunia, "Mero: A statistical approach for hardware trojan detection," in *International Workshop on Cryptographic Hardware and Embedded Systems*, 2009, pp. 396–410.

[12] Y. Lyu and P. Mishra, "Scalable activation of rare triggers in hardware trojans by repeated maximal clique sampling," *IEEE Transactions on CAD*, 2020.

[13] J. Zhang and Q. Xu, "On hardware trojan design and implementation at register-transfer level," in *2013 IEEE international symposium on hardware-oriented security and trust (HOST)*. IEEE, 2013, pp. 107–112.

[14] N. G. Tsoutsos, C. Konstantinou, and M. Maniatakos, "Advanced techniques for designing stealthy hardware trojans," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2014, pp. 1–4.

[15] N. Fern, I. San, C. K. Koç, and K.-T. T. Cheng, "Hiding hardware trojan communication channels in partially specified soc bus functionality," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 9, pp. 1435–1444, 2016.

[16] Y. Jin, N. Kupp, and Y. Makris, "Experiences in hardware trojan design and implementation," in *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*. IEEE, 2009, pp. 50–57.

[17] T. Reece, D. B. Limbrick, X. Wang, B. T. Kiddie, and W. H. Robinson, "Stealth assessment of hardware trojans in a microcontroller," in *2012 IEEE 30th International Conference on Computer Design (ICCD)*. IEEE, 2012, pp. 139–142.

[18] C. Sturton, M. Hicks, D. Wagner, and S. T. King, "Defeating uci: Building stealthy and malicious hardware," in *2011 IEEE symposium on security and privacy*. IEEE, 2011, pp. 64–77.

[19] M. Xue, C. Gu, W. Liu, S. Yu, and M. O'Neill, "Ten years of hardware trojans: a survey from the attacker's perspective," *IET Computers & Digital Techniques*, vol. 14, no. 6, pp. 231–246, 2020.

[20] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy hardware: Identifying and classifying hardware trojans," *Computer*, vol. 43, no. 10, pp. 39–46, 2010.

[21] M. Rathmair, F. Schupfer, and C. Krieg, "Applied formal methods for hardware trojan detection," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2014, pp. 169–172.

[22] N. Fern, I. San, and K.-T. T. Cheng, "Detecting hardware trojans in unspecified functionality through solving satisfiability problems," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 598–504.

[23] C. Bao, Y. Xie, Y. Liu, and A. Srivastava, "Reverse engineering-based hardware trojan detection," *The Hardware Trojan War: Attacks, Myths, and Defenses*, p. 269, 2017.

[24] N. Vashistha, H. Lu, Q. Shi, D. L. Woodard, N. Asadizanjani, and M. Tehranipoor, "Detecting hardware trojans using combined self testing and imaging," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[25] R. Torrance and D. James, "The state-of-the-art in semiconductor reverse engineering," in *Proceedings of the 48th Design Automation Conference*, 2011, pp. 333–338.

[26] A. R. Waite, Y. Patel, J. J. Kelley, J. H. Scholl, J. Baur, A. Kimura, E. D. Udelhoven, G. D. Via, R. Ott, and D. L. Brooks, "Preparation, imaging, and design extraction of the front-end-of-line and middle-of-line in a 14 nm node finfet device," in *2021 IEEE Physical Assurance and Inspection of Electronics (PAINE)*. IEEE, 2021, pp. 1–6.

[27] T. Krachenfels, J.-P. Seifert, and S. Tajik, "Trojan awakener: detecting dormant malicious hardware using laser logic state imaging (extended version)," *Journal of Cryptographic Engineering*, vol. 13, no. 4, pp. 485–499, 2023.

[28] Taiwan Semiconductor Manufacturing Company Limited, "Second-quarter 2025 earnings conference call transcript," July 2025, n3 reported as 24% of wafer revenue; accessed 5 August 2025.

[29] Samsung Electronics Co. Ltd., "Samsung begins chip production using 3 nm process technology with gaa architecture," Press release, June 2022, accessed 5 August 2025.

[30] J. Cruz, Y. Huang, P. Mishra, and S. Bhunia, "An automated configurable trojan insertion framework for dynamic trust benchmarks," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1598–1603.

[31] I. H. Abbassi, F. Khalid, S. Rehman, A. M. Kamboh, A. Jantsch, S. Garg, and M. Shafique, "Trojanzero: Switching activity-aware design of undetectable hardware trojans with zero power and area footprint," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 914–919.

[32] I. Pomeranz and S. M. Reddy, "A measure of quality for n-detection test sets," *IEEE Trans. on Computers*, vol. 53, no. 11, pp. 1497–1503, 2004.

[33] S. Saha, R. S. Chakraborty, S. S. Nuthakki, D. Mukhopadhyay *et al.*, "Improved test pattern generation for hardware trojan detection using genetic algorithm and boolean satisfiability," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2015, pp. 577–596.

[34] Z. Pan and P. Mishra, "Automated test generation for hardware trojan detection using reinforcement learning," in *Asia and South Pacific Design Automation Conference*, 2021, pp. 408–413.

[35] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 137–143.

[36] Y. Xie and A. Srivastava, "Mitigating sat attack on logic locking," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2016, pp. 127–146.

[37] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu, "Sarlock: Sat attack resistant logic locking," in *Hardware Oriented Security and Trust (HOST), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 236–241.

[38] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1601–1618.

[39] Y. Liu, M. Zuzak, Y. Xie, A. Chakraborty, and A. Srivastava, "Robust and attack resilient logic locking with a high application-level impact," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 17, no. 3, pp. 1–22, 2021.

[40] B. Shakya, X. Xu, M. Tehranipoor, and D. Forte, "Cas-lock: A security-corruptibility trade-off resilient logic locking scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 175–202, 2020.

[41] Y. Liu, M. Zuzak, Y. Xie, A. Chakraborty, and A. Srivastava, "Strong anti-sat: Secure and effective logic locking," in *2020 21st International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2020, pp. 199–205.

[42] N. Creignou and M. Hermann, "Complexity of generalized satisfiability counting problems," *Information and computation*, vol. 125, no. 1, pp. 1–12, 1996.

[43] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Security analysis of anti-sat," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 342–347.

[44] N. Mohyuddin, E. Pakbaznia, and M. Pedram, "Probabilistic error propagation in a logic circuit using the boolean difference calculus," in *Advanced Techniques in Logic Synthesis, Optimizations and Applications*. Springer, 2011, pp. 359–381.

[45] D. Sirone and P. Subramanyan, "Functional analysis attacks on logic locking," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2514–2527, 2020.

[46] Z. Han, M. Yasin, and J. J. Rajendran, "Does logic locking work with {EDA} tools?" in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1055–1072.

Yuntao Liu is an Assistant Professor in the Department of Electrical and Computer Engineering at Lehigh University. His research interests include hardware supply chain security, hardware security for artificial intelligence, side-channel analysis, quantum computing security, and biocomputing modeling and security. He received his PhD in Electrical Engineering from the University of Maryland, College Park in 2020.

Aruna Jayasena is an Assistant Professor in the Department of Computer Science and Engineering at the University of Tennessee at Chattanooga. His research interests include systems security, hardware-firmware co-validation, applied cryptography, trusted execution, side-channel analysis, and test generation. He received his PhD in Computer Engineering from the University of Florida in 2025.

Prabhat Mishra is a Professor in the Department of Computer and Information Science and Engineering at the University of Florida. His research interests include embedded systems, hardware security, trustworthy AI, and quantum computing. He currently serves as an Associate Editor of ACM Transactions on Design Automation of Electronic Systems and ACM Transactions on Embedded Computing Systems. He is an IEEE Fellow, an AAAS Fellow, and an ACM Distinguished Scientist.

Ankur Srivastava is the Inaugural Director of Semiconductor Initiatives and Innovation at the University of Maryland. Prior to that he was the seventh director of the Institute for Systems Research (ISR). He has a joint appointment in the Electrical and Computer Engineering Department and ISR. Dr. Srivastava received his B.Tech in Electrical Engineering from Indian Institute of Technology Delhi in 1998 and PhD in Computer Science from UCLA in 2002. He was awarded the prestigious Outstanding Dissertation Award from the CS department of UCLA in 2002.