2021 IFIP/IEEE 29th International Conference on Very Large Scale Integration (VLSI-SoC) | 978-1-6654-2614-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/VLSI-SoC53125.2021.9606979

Logic Locking at the Frontiers of Machine Learning: A Survey on Developments and Opportunities

Dominik Sisejkovic, Lennart M. Reimann, Elmira Moussavi, Farhad Merchant, and Rainer Leupers
Institute for Communication Technologies and Embedded Systems, RWTH Aachen University, Germany
{sisejkovic, reimannl, moussavi, merchantf, leupers}@ice.rwth-aachen.de

***Abstract*—In the past decade, a lot of progress has been made in the design and evaluation of logic locking; a premier technique to safeguard the integrity of integrated circuits throughout the electronics supply chain. However, the widespread proliferation of machine learning has recently introduced a new pathway to evaluating logic locking schemes. This paper summarizes the recent developments in logic locking attacks and countermeasures at the frontiers of contemporary machine learning models. Based on the presented work, the key takeaways, opportunities, and challenges are highlighted to offer recommendations for the design of next-generation logic locking.**

***Index Terms*—logic locking, machine learning, neural networks, reverse engineering, genetic algorithms**

I. Introduction

The involvement of untrusted parties in the modern Integrated Circuit (IC) supply has given rise to a plethora of security concerns, including Intellectual Property (IP) piracy, reverse engineering, counterfeiting, and hardware Trojans [1], [2]. Consequently, a variety of countermeasures have been introduced, including IC metering [3], split manufacturing [4], camouflaging [5], and logic locking [6]. Among these, only logic locking can protect a design against all untrusted parties in the supply chain [6], [7].

A. Logic Locking: A Brief Overview

Logic locking performs design manipulations by binding the correct functionality of a hardware design to a secret key that is only known to the legitimate IP owner. Hereby, both the original functionality and the structure of the design remain concealed while passing through the hands of external design houses and the foundry. In the past decade, various security aspects of logic locking have been thoroughly evaluated through the introduction of key-recovery attacks [8], [9], among which the Boolean satisfiability (SAT) attack has gained a lot of attention [10]. This has led to a division of logic locking into *pre- and post-SAT schemes*. Pre-SAT schemes were focusing on specific security features, such as random XOR/XNOR key-gate insertion [11], thwarting the path-sensitization attack [12] or maximizing output corruption for incorrect keys [13]. With the introduction of SAT-based attacks, the design objective has shifted towards achieving SAT-resilience, resulting in a new generation of schemes, including SARLock [14], Anti-SAT [15], CASLock [16], SFLL [17], and others [8].

978-1-6654-2614-5/21/$31.00 ©2021 IEEE

B. The Advent of Machine Learning

With the advent of efficient and easy-to-use Machine Learning (ML) models, ML-based techniques have been gradually introduced into various hardware-security domains [18], [19]. The latest efforts in the logic locking community have been invested in challenging the security properties of locking schemes using ML. Recent works were able to efficiently attack pre- and post-SAT schemes [20]–[28]. The introduction of ML-based tools for the security analysis of logic locking *has opened up a new chapter* in the design of locking schemes and attacks, thereby initiating the start of the *post-ML* locking-scheme era. Herewith, the ML ecosystem offers a novel path to uncover hidden vulnerabilities and provide new directions in the development of future ML-resilient locking schemes.

Contributions The ML era has undoubtedly initiated a new stage in logic locking design and evaluation. In this paper, we review all major developments in the domain of *ML-based attacks and countermeasures in logic locking*, and analyze major challenges and research opportunities. Note that a comprehensive overview of the state of pre-ML schemes and attacks can be found in [6]–[9], [29].

The rest of this paper is organized as follows. Section II introduces the relevant background on logic locking. Section III reviews the major developments in ML-based logic locking attacks and compiles a summary of the open challenges and opportunities. Finally, Section IV concludes the paper.

II. Background

This section introduces the preliminaries on classification, working principles and attack models of logic locking.

A. Classification

Logic locking can be generally classified into two orthogonal classes: *combinational* and *sequential* [29]. Combinational logic locking performs key-dependent manipulations in the combinational path of a design. On the other hand, sequential logic locking focuses on transforming and obfuscating the state space of a circuit. As the reviewed work operates in the domain of combinational locking, in the rest of this work, the term *logic locking* refers to combinational locking schemes.

B. Working Principles

The idea of logic locking lies in the functional and structural manipulation of a hardware design that creates a dependency to an activation key, hereby trading area, power, and delay

Authorized licensed use limited to: Indian Institute of Information Technology Kottayam. Downloaded on October 09,2024 at 09:22:51 UTC from IEEE Xplore. Restrictions apply.

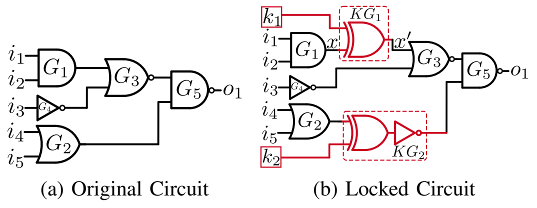

Fig. 1. Example: logic locking using XOR/XNOR gates.

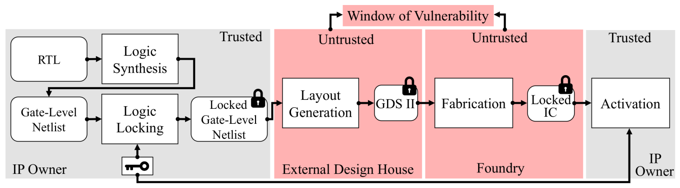

Fig. 2. Logic locking in the IC design and fabrication flow.

for security. If the correct key is provided, the locked design will perform as originally intended *for all input patterns*. Otherwise, an incorrect key will yield an incorrect output for at least *some input patterns*. Logic locking can be performed on different design levels. However, typically logic locking is deployed on a gate-level netlist through the insertion of additional gates (known as *key gates*) or more complex structures. A visual example of a locked design is shown in Fig. 1 (b). Here, the original netlist in Fig. 1 (a) is locked through the insertion of two key-controlled gates in the form of an XOR and XNOR (XOR + INV) gate, marked as KG_1 and KG_2, respectively. To understand the functional implications of the key gates, let us consider the gate KG_1. This key gate takes two inputs: the original wire x (output of gate G_1) and the input key bit k_1. If a correct key value is set, i.e., $k_1 = 0$, the value of x is preserved and forwarded to x'. However, if an incorrect key value is set, i.e., $k_1 = 1$, the value of x is inverted, leading to incorrect output values. Based on this concept, throughout the past decade, a variety of locking schemes have been introduced, based on XOR, XNOR, AND, OR, and MUX gates as well as more elaborate structures [6].

C. Logic Locking in the IC Supply Chain

The role of logic locking in the IC supply chain is demonstrated in Fig. 2. Based on a trusted Register-Transfer Level (RTL) design, the legitimate IP owner performs logic synthesis to generate a gate-level netlist. At this point, logic locking is deployed, resulting in a locked netlist and an activation key. Typically, after the netlist is locked, another synthesis round is performed to facilitate the structural integration of scheme-induced netlist changes. Therefore, we can differentiate between the pre-resynthesis and post-resynthesis netlist. The former is locked but not resynthesized, while the latter is locked and resynthesized. In the next step, the locked netlist proceeds in the untrusted part of the supply chain. This often includes an untrusted external design house (for layout synthesis) and the foundry. After fabrication, the produced IC is returned to the IP owner for activation. Herewith, logic locking protects a design by concealing its functional and structural secrets in the activation key, thereby bridging the untrusted regime gap. In terms of hardware Trojans, it is assumed that a sound understanding of the design's functionality and structure is required to insert an intelligible, controllable and design-specific Trojan (e.g, a targeted denial-of-service attack). However, functionality-independent Trojans remain viable. These include, e.g., the manipulation of the circuit's physical characteristics, leading to performance or reliability degradation. In the former case, finding the activation key is a prerequisite for successfully performing the reverse-engineering process.

D. Attack Model

The attack model includes the following: (i) the attacker has access to the locked netlist, either as an untrusted design house or by reverse engineering the layout, (ii) the location of the key inputs (pins) is known, (iii) the deployed locking scheme is known, and (iv) the attacker has access to an activated IC to use as oracle for retrieving golden Input/Output (I/O) patterns.

The fourth assumption is the differentiating factor that classifies all key-retrieval attacks into *oracle-less* and *oracle-guided* attacks. Oracle-less attacks assume that an activated design is not available. This is often the case in low-volume production for security-critical applications [30]. On the contrary, the oracle-guided scenario assumes the availability of an activated IC, thereby representing a high-volume production setting [31]. Moreover, sometimes, an attack assumes the availability of only a few golden I/O patterns, e.g., in the form of test vectors. Due to the available knowledge that is provided by these patterns, these attacks fall into the oracle-guided class. As discussed in the next section, ML-based attacks have been explored in both classes. Henceforth, we refer to the gate-level netlist under attack as the *target* netlist.

III. Logic Locking in the Machine Learning Era

This section describes the recent developments of ML-based applications in the logic locking domain.

A. ML-Based Attacks

Previous work in this domain has mostly been focusing on the development of novel ML-based attacks on logic locking, both for the oracle-less and oracle-guided model. A simplified visualization of all reviewed attack flows is presented in Fig. 3. Hereby, the attacks are presented in the *deployment* stage (after training). An exhaustive comparison of the attacks is summarized in Table II. The comparison lists the attacks in order of appearance per attack class. For convenience, a glossary of the used acronyms is given in Table I.

The following review reflects the descriptions in Fig. 3 and Table II, thereby only focusing on the *basic attack mechanisms*. All other details can be found in the mentioned *comparison table* and the provided references.

1) Oracle-Less Attacks: ML-based attacks in this class have exploited *scheme-related structural residue* to identify a correct key-bit value or the locking circuitry itself. This category includes SAIL [22], SnapShot [21], and GNNUnlock [20].

Authorized licensed use limited to: Indian Institute of Information Technology Kottayam. Downloaded on October 09,2024 at 09:22:51 UTC from IEEE Xplore. Restrictions apply.

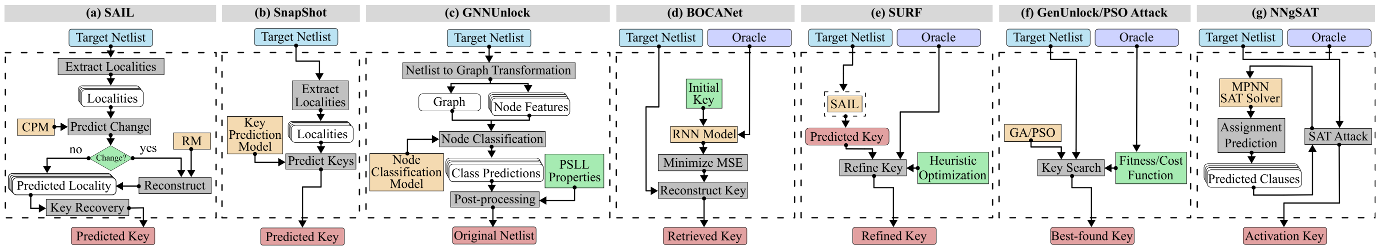

Fig. 3. ML-based attacks on logic locking (deployment phase).

SAIL: This attack deploys ML algorithms to retrieve local logic structures from the post-resynthesis target netlist to predict the correct key values (Fig. 3 (a)). The attack targets XOR/XNOR-based logic locking exclusively. Hereby, the attack exploits two leakage points of the locking flow: (i) the deterministic structural changes induced by logic synthesis around the key-affected netlist gates and (ii) the nature of XOR/XNOR-based locking (XOR for key bit 0 and XNOR for key bit 1). Therefore, the attack encloses two components: the Change Prediction Model (CPM) and the Reconstruction Model (RM). For each netlist subgraph around a key input (known as *locality*), CPM is trained to predict whether a synthesis-induced change has occurred. If a change is predicted, RM is deployed to reconstruct the pre-resynthesis netlist structures, i.e., it reverses the effect of the synthesis process. Finally, based on the intrinsic nature of XOR/XNOR-based locking, SAIL extracts the correct key value.

SnapShot: The attack utilizes a neuroevolutionary approach to automatically design suitable neural networks to *directly* predict correct key values from a locked post-resynthesis target netlist (Fig. 3 (b)). The attack exploits the structural alterations induced by locking schemes to learn a correlation between the key-induced structural patterns and the key value. Compared to SAIL, SnapShot implements an end-to-end ML approach, thereby having the advantage of being applicable to any locking scheme as well as not relying on learning specific transformation rules of the logic synthesis process.

GNNUnlock: The attack leverages a graph neural network to identify all the gates in a post-resynthesis target netlist that belong to the locking circuitry (Fig. 3 (c)). Therefore, compared to SAIL and SnapShot, GNNUnlock learns to differentiate locking gates from regular gates instead of learning a correlation between the netlist sub-graphs and the correct key. To enhance the removal accuracy, after identification, a deterministic post-processing mechanism is deployed to remove the gates depending on the intrinsic features of the underlying schemes. Hereby, GNNUnlock has specifically been trained and deployed to target SAT-attack-resilient schemes, i.e., Provably Secure Logic Locking (PSLL). Herein lies the success of the attack: PSLLs often induce isolable structural netlist changes to produce a specific SAT-resilient circuit behavior. In the pre-ML era, it has been long assumed that PSLLs can be protected through additional locking and resynthesis.

2) Oracle-Guided Attacks: Due to the availability of an oracle, existing ML-based attacks in this class have mostly exploited functional features of the target IC. This class includes the following attacks: BOCANet [26], [39], SURF [27], GenUnlock [23], NNgSAT [24], and the PSO-guided attack [25].

BOCANet: This attack leverages Recurrent Neural Networks (RNN) based on long short-term memory to construct a correct activation key (Fig. 3 (d)). The ML model is trained on a sequence of I/O observations taken from an activated IC, thereby learning the functional I/O mapping of the circuit, i.e., its Boolean function. Once trained, the key retrieval consists of two steps. First, a random key is applied to the model as input. Second, the initial key value is subsequently updated based on the Mean-Squared-Error (MSE) of the trained outputs and the newly generated outputs that are affected by the introduced key. Note that the ML model can be utilized to predict correct inputs or outputs as well. BOCANet exploits the functional effect a correct key has on generating a correct I/O mapping.

SURF: This attack is based on a joint structural and functional analysis of the circuit to retrieve the activation key (Fig. 3 (e)). The ML-aspect of the attack lies in *utilizing the SAIL attack* to generate the pre-resynthesis netlist structures and a seed key. Afterwards, based on the outputs of SAIL, SURF iteratively refines the key by means of a structure-aware greedy optimization algorithm guided by a functional simulation of the obfuscated netlist and a set of golden I/O pairs. The optimization is guided by the observation that specific key gates only affect a specific set of outputs. Thus, the key bits can be partitioned based on which output they affect. Performing a systematic perturbation of the key bits can lead to a more refined key. The success of the heuristic is grounded in the limited local effects that traditional locking

TABLE I
GLOSSARY

Acronym	Definition	Acronym	Definition
AND-OR	AND/OR-based LL [32]	LUT	Lookup table
Anti-SAT	Anti-Boolean satisfiability [15]	MLP	Multi-layer perceptron
C	Combinational circuits	MPNN	Message-passing neural network
CNN	Convolutional neural network	OG	Oracle-guided attack
CS	Logic-cone-size-based LL [33]	OL	Oracle-less attack
D-RNN	Deep recurrent neural network	OLL	Optimal LL [34]
FLL	Fault analysis-based LL [13]	PSO	Particle swarm optimization
FU	Functional	RLL	Random LL [11]
GA	Genetic algorithm	S	Sequential circuits
GL	Gate level	SFLL-HD	Stripped functionality LL [17]
GNN	Graph neural network	SLL	Strong (secure) LL [12]
LL	Logic locking	ST	Structural
LSTM	Long short-term memory	TTLock	Tenacious and traceless LL [35]

Authorized licensed use limited to: Indian Institute of Information Technology Kottayam. Downloaded on October 09,2024 at 09:22:51 UTC from IEEE Xplore. Restrictions apply.

TABLE II
OVERVIEW OF ML-BASED ATTACKS ON LOGIC LOCKING

Attack	Objective	Class	Level/IC Type/Attack Basis	ML Model	Benchmarks[16]	Evaluated Schemes	Scheme Inde-pendent	Exact Output[15]	Evaluate Key Length in Bits	% Accuracy [min, max]	Time Complexity[17]	Known Protection
SAIL [22]	key retrieval	OL	GL/C,S[1]/ST	Random Forest	ISCAS'85	RLL, SLL, CS	✗	✗	$\{4, \cdots, 192\}$[4]	$[66.89, 73.88]$	$\mathcal{O}(l)$[13]	UNSAIL [36], SARO [37][22]
SnapShot [21]	(i) GSS and (ii) SRS key retrieval[14]	OL	GL/C,S/ST	MLP, CNN, GA	ISCAS'85, ITC'99, Ariane RV	RLL	✓	✗	$\{64\}$	$[57.71, 61.56]^{(i)}$, $[71.57, 81.67]^{(ii)}$	$\mathcal{O}(l)$[13]	D-MUX [38]
GNNUnlock [20]	key gates removal	OL	GL/C,S/ST	GNN	ISCAS'85, ITC'99	Anti-SAT, TTLock, SFLL-HD	✗[12]	✗	$\{8, \cdots, 128\}$	$[100.00]$[11]	$\mathcal{O}(n)$[13]	●
BOCANet [26], [39]	key retrieval	OG[2]	GL/C/FU	D-RNN & LSTM	Trust-Hub, ISCAS'85	RLL	✓	✗	$\{32, 64, 128, 256\}$	$[89.00, 100.00]$	$\mathcal{O}(\alpha)$[19]	●
SURF [27]	key retrieval	OG	GL/C,S[1]/ST,FU	SAIL & heuristic optimiza-tion	ISCAS'85	RLL, SLL, CS	✗[3]	✗	$\{4, \cdots, 192\}$[4]	$[90.58, 98.83]$	$\mathcal{O}(t)$[13]	UNSAIL [36]
GenUnlock [23]	key retrieval	OG	GL/C,S/FU	GA	ISCAS'85, MCNC	RLL,SLL, AND-OR, FLL	✓	✗	$\{8, \cdots, 1618\}$[7]	$[100.00]$[8]	$\mathcal{O}(\beta)$[20]	●
NNgSAT [24]	key retrieval	OG	GL/C,S/FU	MPNN	ISCAS'85, ITC'99	SAT-hard[5]	✓	✓	n/a[5]	$[93.50]$[6]	$\mathcal{O}(\lambda)$[18]	●
PSO Attack [25]	key retrieval	OG[9]	GL/C,S/FU	PSO	ISCAS'85, ITC'99	RLL, OLL	✓	✗	$\{64, 128\}$	$[82.07, 99.80]$[10]	$\mathcal{O}(\gamma)$[21]	point-function locking [15], [17]

[1] The attack is in theory applicable to sequential circuits, however no evaluation has been performed yet.
[2] The attack relies on having access to at least some golden input/output patterns ($< 0.5\%$ of total I/O pairs).
[3] In theory, the key-refinement search algorithm could be utilized based on any seed key. However, this has not been addresses thus far.
[4] The following key lengths have been evaluated: $\{4, 8, 16, 24, 32, 64, 96, 128, 192\}$.
[5] $n \times m$ bitwise multipliers ($8 < m, n < 32$), $n \times m$ crossbar network of 2-to-1 MUXes ($16 < m, n < 36$), n-input LUTs built by 2-to-1 MUXes ($n < 16$), and n-to-1 AND-trees.
[6] Indicates the percentage of successfully de-obfuscated circuits compared to the baseline [10].
[7] The key length is selected based on an area overhead of 5%, 10% or 25%.
[8] The quality of the retrieved approximate keys is quantified by a user-defined output-fidelity measure.
[9] The attack relies on an oracle without access to the scan chain.
[10] Accuracy refers to the average number of cases where the retrieved key results in 0% erroneous outputs for 10^6 random patterns.
[11] The accuracy refers to the successful removal of the locking circuitry (not the key retrieval).
[12] The attack has not been evaluated for other locking schemes so far and the post-processing steps are scheme-specific.
[13] Notation: the key length l, the number of netlist nodes n, the total number of iterations $t = p \cdot n + p \cdot w + r \cdot l \cdot i \cdot n + r \cdot l \cdot i \cdot p$, where p is the number of output pins, i is the number of IO pairs, r is the number of runs, and w is the number of wires.
[14] GSS refers to the generalized set scenario which trains the ML model based on a set of locked benchmarks that are different from the target. SRS captures the self-referencing scenario where the training data is generated by re-locking the target benchmark.
[15] If the output of the attack is an exact result, the attack can guarantee a 100% correct deobfuscation for the complete I/O space.
[16] ISCAS'85 [40], MCNC [41], ITC'99 [42], RISC-V Ariane core [43], and Trust-Hub [44], [45].
[17] If the time complexity can be clearly determined, it refers to the time complexity after the training process.
[18] The execution time of the SAT attack is $\sum_{i=1}^{\lambda} t_i$, where λ is the number of iterations and t_i the time required for one SAT-solver call. λ depends on the characteristics of the search space and the branching preferences of the SAT solver. Therefore, the time complexity of the attack is typically measured in terms of λ. More details can be found in [46].
[19] The complexity is linear to the number of training samples α, as the final key is determined based on the MSE of the trained outputs and the key-induced generated outputs.
[20] $\beta = g \cdot p \cdot l$, where g is the number of generations, p is the population size, and l is the key length. Note that the complexity changes with any adaptations of the GA.
[21] $\gamma = g \cdot p \cdot \delta$, where g is the number of generations, p is the population size, and δ the complexity of performing circuit simulation for the fitness evaluation.
[22] An empirical evaluation of the resilience against SAIL has not been presented in the paper; only a discussion based on a proposed metric system has been provided.

schemes have on the value of the output for incorrect keys.

GenUnlock: The attack flow of GenUnlock leverages a Genetic Algorithm (GA)-based exploration of suitable activation keys for a locked circuit (Fig. 3 (f)). The heuristic search is steered by the key fitness that is computed based on the matching ratio of the key on the golden I/O training set. Through multiple generations, the fitness of the key population is subsequently improved through the application of genetic operators (selection, crossover, and mutation). Once the accepted tolerance for the correctness of the key is reached, the algorithm returns the set of the fittest keys. Similarly to SURF, GenUnlock exploits the fact that a heuristic key-refinement procedure eventually leads to more accurate activation keys.

NNgSAT: The main objective of NNgSAT is the deployment of a Message-Passing Neural Network (MPNN) to facilitate the resolution of SAT-hard circuit structures during the application of a SAT-based attack (Fig. 3 (g)). The motivation is driven by the fact that common SAT solvers run into scalability issues when tackling hard-to-be-solved locked circuit structures, e.g, multipliers and AND trees. Therefore, in NNgSAT, a neural network is trained to predict the satisfying assignment on a set of SAT-hard cases. In deployment, the SAT-attack flow offloads the SAT-hard problems to the trained model to speed up the attack procedure. The effectiveness of NNgSAT lies in the fact that it is possible to transfer prediction knowledge from learned clauses to unseen problems.

PSO-guided Attack: This attack is based on a Particle Swarm Optimization (PSO) heuristic that searches through the key space directed by a selected cost function (Fig. 3 (f)). The cost function is modeled as the Hamming distance between the golden and the obtained output responses (for a selected key). Therefore, the search algorithm relies on having access to an activated IC to compare against. A major motivator for this attack is its applicability without having access to an open scan chain, as this is often a limiting factor for SAT-based attacks. In essence, the PSO-guided attack and GenUnlock are similar

Authorized licensed use limited to: Indian Institute of Information Technology Kottayam. Downloaded on October 09,2024 at 09:22:51 UTC from IEEE Xplore. Restrictions apply.

in nature, as both rely on black-box evolutionary procedures guided by a functionality-driven objective function.

B. ML-Resilient Schemes

UNSAIL: This logic locking scheme has been developed to thwart attacks that target the resolution of structural transformations of logic synthesis [36]. The core idea of UNSAIL is to generate confusing training data that leads to false predictions in the CPM and RM modules of SAIL. This is realized through the additional manipulation of the netlist after synthesis to force the existence of *equivalent* netlist sub-graph observations that are linked to *different* key values.

SARO: The Scalable Attack-Resistant Obfuscation (SARO) operates in two steps [37]. First, SARO splits the design into smaller partitions to maximize the structural alterations in the netlist. Second, a systematic truth table transformation is deployed to lock the partitions. In order to increase the complexity of pattern recognition attacks (such as SAIL), the transformations aim to maximize randomness in the netlist.

Point-Functions and PSO: As mentioned in Table II, the PSO-guided attack is not applicable to point-function-based locking schemes. The reason is that this type of locking yields SAT-resilient behavior in which any incorrect key corrupts only *a very limited amount of outputs*. Consequently, this behavior offers no advantage in the guidance of the heuristic search, as it does not yield differentiating fitness values. Note that the applicability of point functions depends on the design of the fitness function that is used to guide the heuristic.

D-MUX: The recently introduced Deceptive Multiplexer (D-MUX) LL scheme builds on the concept of multiple MUX-based insertion strategies that create structural paths that are equally likely to be driven by 0 or 1 key values [38]. Hence, D-MUX offers efficient protection against data-driven attacks.

C. Other Applications of ML

Deobfuscation Runtime Prediction: Apart from ML-based attacks, machine learning has also found its way into other aspects of logic locking. A recent work has designed a framework named ICNet for the prediction of the key-retrieval runtime for SAT-based attacks using graph neural networks [28]. The framework obtains the predicted deobfuscation runtime based on the characteristics of the circuit topology. ICNet offers an end-to-end approach to evaluate the hardness of logic locking with respect to the SAT attack, thereby increasing the development efficiency of novel locking schemes.

ML-Attack on Analog IC Locking: ML has started to have an impact on locking mechanisms even beyond digital circuits. The authors in [47] have developed an oracle-guided attack on locked analog ICs using genetic algorithms. The approach has successfully broken all known analog logic locking techniques.

D. Lessons Learned - Challenges and Opportunities

The presented efforts gather around two focal points: oracle-less and oracle-guided attacks. The intrinsic mechanisms of these attacks shed light on major vulnerabilities in existing logic locking that are exploitable by ML. We summarize the observations as follows:

1) Structural vs. Functional Analysis: Oracle-guided attacks focus on functional aspects of schemes, whereas structural leakage is exploited in the oracle-less model due to the absence of an activated IC. This indicates two pitfalls. First, the evaluated schemes have a predictable effect on the functionality of the circuit for incorrect keys, enabling the possibility to perform a guided heuristic search of the key space. Second, the existing schemes induce structural changes which strongly correlate with the value of the key. A mitigation depends on what an attempted attack tries to exploit. For example, to overcome SAIL or SnapShot, a scheme must not reveal anything about the correctness of the key through the induced structural change. This is achieved if the inserted change does not differ depending on the key value. Similarly, to protect against GNNUnlock, the deployed schemes have to ensure resilience against isolation, i.e., the locking circuitry must not be structurally independent from the original design. In the case of GenUnlock and the PSO attack, the behavior of the underlying scheme must be constant or fully random for all incorrect key values; disabling any chance for a guided heuristic convergence towards a correct key. Similar observations can be made for the other attacks as well. Nevertheless, conveying all necessary security objectives into a uniform locking scheme remains an open challenge.

2) Logic Synthesis for Security: The reliance on logic synthesis transformations to enable the security of logic locking schemes has to be revised. As shown in SAIL, the synthesis rules are predictable and reversible, thereby having little impact on deducing a correct key for traditional schemes. Nevertheless, resynthesis can increase the difficulty to reverse engineering the functionality of a design. However, this needs further evaluation in the context of novel locking policies.

3) Structural Isolation of Post-SAT Schemes: PSLL schemes induce isolable changes which create a clear structural distinction between the locking-related gates and the original (regular) gates. The main reason for this vulnerability is the requirement of achieving SAT-resistant behavior for incorrect keys. This specific functional pattern requires significant structural netlist changes consolidated in isolable components (e.g., the tree of complementary logic in Anti-SAT or the restore/perturb units in TTLock and SFLL-HD).

4) Output Uncertainty: Regardless of the attack type, ML-based attacks tend to generate an approximate output (as marked in Table II), meaning that the retrieved key or deobfuscated circuit cannot be evaluated as correct with an exact certainty. Note that this is not the case for NNgSAT. Nevertheless, the approximate output can be utilized as seed for other attacks to speed up the reverse engineering procedure.

5) RTL vs Gate-Level: As shown in Table II, the existing attacks focus on extrapolating information leakage from gate-level netlists. So far, it is unclear if the same leakage is present in RTL-based locking schemes as well.

6) Overspecialization: Available mitigation schemes suffer from overspecialization for thwarting specific attack vectors. So far, it has not been evaluated which form of logic locking offers a comprehensive protection against any form of ML-

Authorized licensed use limited to: Indian Institute of Information Technology Kottayam. Downloaded on October 09,2024 at 09:22:51 UTC from IEEE Xplore. Restrictions apply.

based guessing attack. Hence, a potentially fruitful opportunity lies within the automatic design of resilient schemes [48].

7) ML for Sequential Locking: To the best of our knowledge, ML-based techniques have not been deployed yet for the evaluation of sequential logic locking techniques.

8) The Need for Benchmarks: There is still a need to have access to a rich benchmark set for data-driven analysis [49].

IV. Conclusion

This work summarizes the recent developments of logic locking at the frontiers of machine learning. The presented ML-based attacks indicate the presence of structural and functional leakage in contemporary locking schemes which has been overlooked by the traditional interpretation of security. We show that an ML-based analysis is able to pinpoint novel vulnerabilities and challenge existing security assumptions. The offered discussion consolidates the major ML-induced challenges in logic locking design, offering a fruitful ground for future research directions in the post-ML era.

References

[1] M. Rostami *et al.*, "A primer on hardware security: Models, methods, and metrics," *Proc. of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[2] K. Xiao *et al.*, "Hardware Trojans: Lessons learned after one decade of research," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, no. 1, May 2016. [Online]. Available: https://doi.org/10.1145/2906147

[3] F. Koushanfar, "Provably secure active IC metering techniques for piracy avoidance and digital rights management," *IEEE TIFS*, vol. 7, no. 1, pp. 51–63, 2012.

[4] J. Rajendran *et al.*, "Is split manufacturing secure?" in *2013 DATE*, 2013, pp. 1259–1264.

[5] J. Rajendran *et al.*, "Security analysis of integrated circuit camouflaging," in *Proceedings of the 2013 ACM SIGSAC CCS*, ser. CCS '13. New York, NY, USA: ACM, 2013, p. 709–720.

[6] M. Yasin *et al.*, "Evolution of logic locking," in *2017 IFIP/IEEE VLSI-SoC*, 2017, pp. 1–6.

[7] A. Chakraborty *et al.*, "Keynote: A disquisition on logic locking," *IEEE TCAD*, vol. 39, no. 10, pp. 1952–1972, 2020.

[8] K. Zamiri Azar *et al.*, "Threats on logic locking: A decade later," in *2019 GLSVLSI*. New York, NY, USA: ACM, 2019, p. 471–476.

[9] M. Yasin *et al.*, "Removal attacks on logic locking and camouflaging techniques," *IEEE Transactions on Emerging Topics in Computing*, vol. 8, no. 2, pp. 517–532, 2020.

[10] P. Subramanyan *et al.*, "Evaluating the security of logic encryption algorithms," in *2015 IEEE HOST*, 2015, pp. 137–143.

[11] J. A. Roy *et al.*, "EPIC: Ending piracy of integrated circuits," in *2008 Design, Automation and Test in Europe*, 2008, pp. 1069–1074.

[12] M. Yasin *et al.*, "On improving the security of logic locking," *IEEE TCAD*, vol. 35, no. 9, pp. 1411–1424, 2016.

[13] J. Rajendran *et al.*, "Fault analysis-based logic encryption," *IEEE Transactions on Computers*, vol. 64, no. 2, pp. 410–424, 2015.

[14] M. Yasin *et al.*, "SARLock: SAT attack resistant logic locking," in *2016 IEEE HOST*, 2016, pp. 236–241.

[15] Y. Xie *et al.*, "Anti-SAT: Mitigating SAT attack on logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 2, pp. 199–207, 2019.

[16] B. Shakya *et al.*, "CAS-lock: A security-corruptibility trade-off resilient logic locking scheme," *IACR TCHES*, vol. 2020, no. 1, Nov. 2019.

[17] M. Yasin *et al.*, "Provably-secure logic locking: From theory to practice," in *Proceedings of the 2017 ACM SIGSAC CCS*, ser. CCS '17. New York, NY, USA: ACM, 2017, p. 1601–1618.

[18] Z. Huang *et al.*, "A survey on machine learning against hardware Trojan attacks: Recent advances and challenges," *IEEE Access*, vol. 8, pp. 10 796–10 826, 2020.

[19] R. Elnaggar *et al.*, "Machine learning for hardware security: Opportunities and risks," *Journal of Electronic Testing*, vol. 34, no. 2, pp. 183–201, Apr 2018.

[20] L. Alrahis *et al.*, "GNNUnlock: Graph neural networks-based oracle-less unlocking scheme for provably secure logic locking," in *IEEE/ACM DATE*, 2021, pp. 780–785.

[21] D. Sisejkovic *et al.*, "Challenging the security of logic locking schemes in the era of deep learning: A neuroevolutionary approach," *J. Emerg. Technol. Comput. Syst.*, vol. 17, no. 3, May 2021.

[22] P. Chakraborty *et al.*, "SAIL: Machine learning guided structural analysis attack on hardware obfuscation," in *2018 AsianHOST*, 2018, pp. 56–61.

[23] H. Chen *et al.*, "GenUnlock: An automated genetic algorithm framework for unlocking logic encryption," in *2019 ICCAD*, 2019, pp. 1–8.

[24] K. Z. Azar *et al.*, "NNgSAT: Neural network guided SAT attack on logic locked complex structures," in *ICCAD 2020*. New York, NY, USA: ACM, 2020.

[25] R. Karmakar *et al.*, "A particle swarm optimization guided approximate key search attack on logic locking in the absence of scan access," in *2020 DATE*, 2020, pp. 448–453.

[26] F. Tehranipoor *et al.*, "Deep RNN-oriented paradigm shift through BOCANet: Broken obfuscated circuit attack," in *2019 GLSVLSI*. New York, NY, USA: ACM, 2019, p. 335–338.

[27] P. Chakraborty *et al.*, "SURF: Joint structural functional attack on logic locking," in *2019 IEEE HOST*, May 2019, pp. 181–190.

[28] Z. Chen *et al.*, "Estimating the circuit de-obfuscation runtime based on graph deep learning," in *2020 DATE*, 2020, pp. 358–363.

[29] K. Shamsi *et al.*, "IP protection and supply chain security through logic obfuscation: A systematic overview," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 24, no. 6, Sep. 2019.

[30] Task Force, "High performance microchip supply," *Annual Report. Defense Technical Information Center (DTIC), USA*, 2005. [Online]. Available: https://apps.dtic.mil/sti/citations/ADA435563

[31] C. Pilato *et al.*, "ASSURE: RTL locking against an untrusted foundry," *IEEE TVLSI*, vol. 29, no. 7, pp. 1306–1318, 2021.

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

[33] S. Amir *et al.*, "Development and evaluation of hardware obfuscation benchmarks," *HaSS*, vol. 2, no. 2, pp. 142–161, Jun 2018. [Online]. Available: https://doi.org/10.1007/s41635-018-0036-3

[34] R. Karmakar *et al.*, "On finding suitable key-gate locations in logic encryption," in *2018 IEEE ISCAS*, 2018, pp. 1–5.

[35] M. Yasin *et al.*, "What to lock? functional and parametric locking," in *GLSVLSI 2017*. New York, NY, USA: ACM, 2017, p. 351–356. [Online]. Available: https://doi.org/10.1145/3060403.3060492

[36] L. Alrahis *et al.*, "UNSAIL: Thwarting oracle-less machine learning attacks on logic locking," *IEEE TIFS*, vol. 16, pp. 2508–2523, 2021.

[37] A. Alaql *et al.*, "Scalable attack-resistant obfuscation of logic circuits," 2020. [Online]. Available: https://arxiv.org/abs/2010.15329

[38] D. Sisejkovic *et al.*, "Deceptive logic locking for hardware integrity protection against machine learning attacks," 2021. [Online]. Available: https://arxiv.org/abs/2107.08695

[39] F. Tehranipoor *et al.*, "Deep RNN-oriented paradigm shift through BOCANet: Broken obfuscated circuit attack," 2018. [Online]. Available: https://arxiv.org/abs/1803.03332

[40] F. Brglez *et al.*, "Combinational profiles of sequential benchmark circuits," in *IEEE ISCAS*, May 1989, pp. 1929–1934 vol.3.

[41] S. Yang, "Logic synthesis and optimization benchmarks version 3.0," *Tech. Report, Microelectronics Centre of North Carolina*, 1991.

[42] F. Corno *et al.*, "RT-level ITC'99 benchmarks and first ATPG results," *IEEE Design & Test of Computers*, vol. 17, no. 3, pp. 44–53, 2000.

[43] F. Zaruba *et al.*, "The cost of application-class processing: Energy and performance analysis of a Linux-ready 1.7-GHz 64-bit RISC-V core in 22-nm FDSOI technology," *IEEE TVLSI*, 2019.

[44] H. Salmani *et al.*, "On design vulnerability analysis and trust benchmarks development," in *2013 IEEE 31st ICCD*, 2013, pp. 471–474.

[45] B. Shakya *et al.*, "Benchmarking of hardware Trojans and maliciously affected circuits," *HaSS*, vol. 1, no. 1, pp. 85–102, 2017.

[46] S. Phatharodom *et al.*, "Modeling SAT-attack search complexity," in *2020 ISCAS*, 2020, pp. 1–5.

[47] R. Y. Acharya *et al.*, "Attack of the genes: Finding keys and parameters of locked analog ICs using genetic algorithm," 2020. [Online]. Available: https://arxiv.org/abs/2003.13904

[48] A. Alaql *et al.*, "LeGO: A learning-guided obfuscation framework for hardware IP protection," *IEEE TCAD*, pp. 1–1, 2021.

[49] S. Amir *et al.*, "Adaptable and divergent synthetic benchmark generation for hardware security," ser. ICCAD '20. NY, USA: ACM, 2020.

Authorized licensed use limited to: Indian Institute of Information Technology Kottayam. Downloaded on October 09,2024 at 09:22:51 UTC from IEEE Xplore. Restrictions apply.