On the Efficacy and Vulnerabilities of Logic Locking in Tree-Based Machine Learning

Brunno Alves de Abreu, *Student Member, IEEE*, Guilherme Paim, *Member, IEEE*, Lilas Alrahis, *Member, IEEE*, Paulo Flores, *Senior Member, IEEE*, Ozgur Sinanoglu, *Senior Member, IEEE*, Sergio Bampi, *Senior Member, IEEE*, and Hussam Amrouch, *Member, IEEE*

***Abstract*— The popularity and widespread usage of machine learning (ML) hardware have created challenges for its intellectual property (IP) protection. Logic locking is a widely used technique for IP protection but has received little attention in error-resilient applications such as ML hardware modules. This work investigates the effectiveness of logic locking when applied to tree-based ML circuits and reveals a critical vulnerability that undermines its effectiveness for single-label ML classifiers. We propose a logic locking scheme to eliminate the vulnerabilities in decision trees (DTs) and random forests (RFs) circuits. In our extensive simulation involving 16 DTs and 16 RFs, our solution consistently thwarts the vulnerability. We further evaluated the security of our approach by considering different obfuscation percentages and launching state-of-the-art oracle-less attacks on logic locking. Our method proves resilient, indicating that by fixing the identified vulnerability, we did not introduce new attack vectors. Further, our investigation indicates that DT/RF accelerators are significantly less vulnerable to oracle-less attacks compared to exact circuits. Overall, our work lays the foundation for future investigations into the effectiveness of logic locking for ML circuits.**

***Index Terms*— Logic locking, machine learning, random forest, decision trees, hardware security.**

Manuscript received 28 February 2024; revised 4 June 2024 and 28 July 2024; accepted 26 August 2024. Date of publication 1 October 2024; date of current version 9 January 2025. This work was supported in part by the ML-DSIV Project funded by CAPES and FCT Research and Development Agencies; in part by the German Research Foundation (DFG) "ACCROSS: Approximate Computing aCROss the System Stack;" and in part by the Center for Cyber Security (CCS), New York University Abu Dhabi (NYUAD). This article was recommended by Associate Editor Y. Tang. *(Brunno Alves de Abreu and Guilherme Paim contributed equally to this work.) (Corresponding author: Guilherme Paim.)*

Brunno Alves de Abreu and Sergio Bampi are with the Graduate Program in Microelectronics (PGMICRO), Institute of Informatics (INF), Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre 91501-970, Brazil (e-mail: baabreu@inf.ufrgs.br; bampi@inf.ufrgs.br).

Guilherme Paim is with the Graduate Program in Microelectronics (PGMICRO), Institute of Informatics (INF), Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre 91501-970, Brazil, and also with the High-Performance Computing Architectures and Systems (HPCAS), Instituto de Engenharia de Sistemas e Computadores–Investigação e Desenvolvimento (INESC-ID), Instituto Superior Técnico, University of Lisbon, 1000-029 Lisbon, Portugal (e-mail: gppaim@inf.ufrgs.br).

Lilas Alrahis is with the Division of Engineering, New York University Abu Dhabi (NYUAD), Abu Dhabi, United Arab Emirates, and also with the Department of Computer and Information Engineering, Khalifa University, Abu Dhabi, United Arab Emirates (e-mail: lilas.malrahis@ku.ac.ae).

Paulo Flores is with the High-Performance Computing Architectures and Systems (HPCAS), Instituto de Engenharia de Sistemas e Computadores–Investigação e Desenvolvimento (INESC-ID), Instituto Superior Técnico, University of Lisbon, 1000-029 Lisbon, Portugal.

Ozgur Sinanoglu is with the Division of Engineering, New York University Abu Dhabi (NYUAD), Abu Dhabi, United Arab Emirates (e-mail: os22@nyu.edu).

Hussam Amrouch is with the Chair of AI Processor Design, Technical University of Munich (TUM), 80333 Munich, Germany, and also with Munich Institute of Robotics and Machine Intelligence (MIRMI), 80992 Munich, Germany (e-mail: amrouch@tum.de).

Digital Object Identifier 10.1109/TCSI.2024.3457541

I. Introduction

MACHINE learning (ML) is rapidly gaining remarkable attention due to its powerful capability to solve a manifold of problems better and/or faster than traditional algorithms and impressively better than humans [1]. To cope with the internet of things (IoT) (i.e, the worldwide interconnected smart things capable of sensing, classifying, actuating, and communicating among themselves and with the environment [2]) service constraints (e.g., energy efficiency, reduced bandwidth, low-latency response, privacy, and security) the ML processing has been shifting from a cloud-centric paradigm towards the extreme edge, also referred to as *TinyML* [3]. This paradigm aims at building efficient ML designs for resource-constrained computing systems mainly powered by batteries [4] as well as self-powered through energy harvesting [5].

While complex and resource-hungry deep neural networks (DNN) pose significant implementation challenges for TinyML, tree-based ML models such as decision trees (DTs) and random forests (RFs) offer cost-efficient alternatives for TinyML [4], [6]. These models require simpler logical operations, making them less resource-intensive. Despite their simplicity, tree-based ML models offer top accuracy in various application domains and are also capable of performing well with limited-dimensional training datasets [7]. Further, they offer improved interpretability compared to DNNs' black-box nature [6].

TinyML has brought ML models to the extreme edge, enabling edge processing and improving the privacy and security of user data. However, this paradigm also distributes ML to billions of edge systems, potentially exposing the models and opening up new challenges for intellectual property (IP) piracy protection. The high costs of developing domain-specific hardware for TinyML further emphasize the urgent need for effective IP piracy protection. To address these concerns, various "design-for-trust" solutions have been proposed by researchers to safeguard the hardware

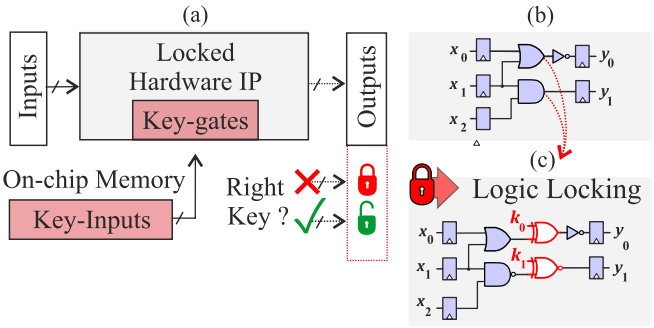

Fig. 1. (a) Locked hardware IP high-level view. (b) Original netlist. (c) Protected netlist using X(N)OR-based logic locking [11].

against supply chain vulnerabilities and hardware-based attacks [8], [9], [10].

Logic locking is a protection technique that obfuscates the structure and functionality of a design using a secret key configuration, as shown in Fig. 1 (a). The original design netlist shown in (b) is locked using X(N)OR gates (referred to as *key-gates*), as shown in (c). The key-gates are controlled by newly added key-inputs k_0 and k_1. A secret key activates the correct functionality of the circuit (i.e., $k_0 = 0$ and $k_1 = 1$), while other key combinations are expected to corrupt the functionality, resulting in an incorrect circuit [11].

The effectiveness of logic locking in protecting hardware IPs has traditionally been evaluated using benchmarks such as ISCAS and ITC [12]. These benchmarks are designed for general-purpose applications that require full exactness to ensure functionality, encompassing areas like arithmetic and logic units, interrupt handlers, communication controllers, cryptography, and microprocessors. In contrast, ML models possess an approximate nature and intrinsic error resiliency, demanding less precision in hardware to yield successful results [13], [14]. This implies that even partially incorrect keys can result in high accuracy for ML models.

In a related exploration [15], the effectiveness of logic locking is investigated within the context of approximate circuits for the first time. The authors demonstrated that an incorrectly unlocked circuit can still generate acceptable results, validated experimentally at both the primitive level (addition and multiplication operations) and in a complete application scenario (neural network inference with approximate multiply-accumulate units). However, no solution was proposed to rectify this issue. In summary, there has been limited research on logic locking for safeguarding ML hardware, posing challenges in terms of cost and effectiveness. Our work addresses these challenges, particularly in the context of logic locking for TinyML systems, as outlined below.

A. Key Research Challenges

1) *Constrained Resources:* (i) Protecting hardware IP on resource-constrained systems in the TinyML domain is challenging due to low overhead requirements. The TinyML domain demands tailored yet effective and secure IP protection solutions. (ii) Moreover, the enhanced efficacy of ML circuits in producing reasonably accurate results, even in error-prone scenarios (i.e., demonstrating error resiliency), poses a challenge to logic locking. This challenge is manifested in the tendency of demanding increased logic locking hardware overhead to safeguard ML IPs, thereby directly impacting point (i). (iii) In addition, existing attacks that reveal part of the logic locking key can lead to a higher demand for hardware overhead to protect ML accuracy, further increasing the challenges of (i) and (ii).
2) *A Lack of Standardized Benchmarks* to study the effectiveness of logic locking on error-resilient hardware [12].
3) *Inadequate Metrics:* Existing logic locking evaluation metrics (e.g., output corruption)[1] may not be suitable for error-resilient circuits used in tree-based ML, leading to a challenge in assessing the security of these systems.
4) *Invalid Function:* Searching for new evaluation metrics, our investigation additionally reveals severe vulnerabilities that undermine the security of logic locking once applied to any kind of ML classifiers, see Fig. 2 (a), which was not discovered by previous work [15]. The output corruption induced by logic locking leads the single-label classifiers to behave in a suspicious manner due to the null or multiple classifications in one inference, allowing an adversary to easily prune out some incorrect key combinations and hence rapidly obtain the secret key. The design solutions against this vulnerability on TinyML domains cannot add much overhead due to constrained resources.

B. Our Novel Concept and Contributions

This work is the first to investigate the cost and effectiveness of logic locking in protecting the design IP of classifiers, considering tree-based ML models such as DTs and RFs. To surmount the above-mentioned challenges:

1) We implement a benchmark suite for tree-based ML hardware classifiers which will be made publicly available (**Section III**). Our benchmark suite contains a set of DTs and RFs comprehending four different datasets (accelAdl [16], activities [17], wearable [18], and wireless [19]) and various tree depths (from 6 to 9), resulting in a total of 32 designs.[2]
2) Then, we investigate the effectiveness of logic locking on DTs and RFs, in terms of inference accuracy drop (i.e., output corruption) (**Section IV**). To this end, we locked the DT and RF netlists using random logic locking (RLL) [11] with logic locking percentages of 5%, 10%, and 25% of the total gate count of each design, resulting in key sizes ranging from $K = 14$ (i.e., 5% in the smaller DT) to $K = 1253$ (i.e., 25% in the largest RF). Locking with such percentages is consistent with

[1]Output corruptibility is a metric to evaluate the effectiveness of logic locking, in which higher corruption indicates higher effectiveness. It measures the number of output bits that are wrong compared with a golden model circuit output (i.e., a circuit with a correct key).

[2]The designs are synthesized to the Silvaco open-source standard-cell library [20], calibrated against measurements from Intel 14nm FinFET.

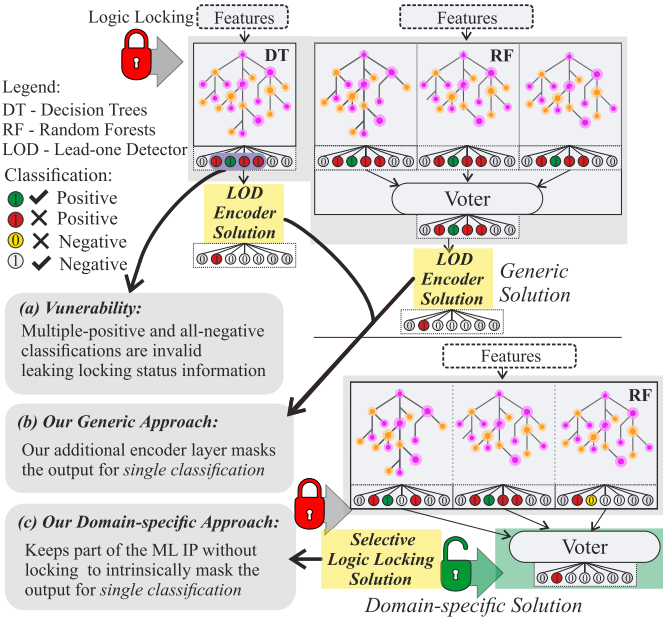

Fig. 2. (a) Traditional logic locking causes single-label ML classifiers to behave in a suspicious manner, leaking secret key information. Our generic approach (b) adds a LOD encoder as a defense layer, masking multi-label classification. Our domain-specific approach (c) applies constrained logic locking, preserving parts of the ML hardware IPs for self-correction.

the state-of-the-art for evaluating the security of logic locking [21]. Depending on the size of the circuit and the locking percentage, the key size changes; for smaller circuits, the key size is smaller, and vice versa.

3) We further evaluate the effectiveness of logic locking when employed to secure both exact and ML circuits, considering different state-of-the-art attacks and threat models. Specifically, we compare the obtained post-attack model accuracy of the locked circuits to the baseline accuracy of the model with the correct key to assess the resiliency of ML circuits in comparison to exact circuits (**Section V**).
4) We propose a novel evaluation metric specific to ML classifiers, i.e., *measuring invalid-output classification*, which aids in identifying vulnerable ML hardware (**Section V**).
5) To overcome the vulnerabilities revealed in this work regarding invalid-output classifications in the one-hot encoding scheme, we propose two approaches (**Section V**); (i) A generic encoder-based approach based on a lead-one detector (LOD) circuit coupled to the IP outputs for redrafting the classification results, ensuring a single class output result while maintaining high output corruption, see Fig. 2 (b). This approach can be applied to any kind of ML classifier, regardless of the underlying hardware implementation. (ii) We further demonstrate how domain knowledge aids in eliminating the above-mentioned vulnerability. Specifically for RFs, we implement a domain-specific approach that constrains locking, to preserve the voter module intact, ensuring a valid output behavior, see Fig. 2 (c).

In summary, our work pushes the field of logic locking forward, allowing for a secure implementation on tree-based ML hardware, which is essential when it comes to edge AI. Our Tree-based ML IP hardware benchmark with logic locking will be open-sourced.[3]

This paper is organized as follows. Section II presents a background on tree-based ML models. Section III presents the benchmarks of the locked DT/RF accelerators developed in this work. Section IV investigates the influence of logic locking on the accuracy of tree-based ML IPs. Moving to Section V, we delve into the security aspects surrounding the use of logic locking in tree-based ML IPs. Initially, we unveil a vulnerability that has the potential to expose key information to the attacker, and subsequently, we propose corrective measures. Furthermore, we conduct state-of-the-art oracle-guided and oracle-less attacks (i.e., considering different attack threat models)[4] on the locked circuits to explore whether the ML accuracy is recovered with a partial key unveiled by such attacks. Section VI concludes the work, highlighting our main findings and contributions.

II. Tree-Based Machine Learning

The tree-based models evaluated in this paper are DTs and RFs, which are described below.

Decision Trees (DTs): are supervised classification and regression ML algorithms. A DT model makes its decisions employing a tree of comparisons between the features of the dataset and a set of thresholds carefully defined during the training phase. The tree is assembled by identifying the most correlated features with the respective target label of the training set, using an information gain metric [23]. This is applied recursively until a halt condition is met (e.g., limitations in the tree depth or when the training set cannot be split anymore). The leaf nodes of the tree define the final decision given a set of feature values.

Random Forests (RFs): are supervised learning algorithms composed of several smaller trees. These models work based on the premise that weaker learners usually achieve better prediction results when compared to a single complex learner. The merge is performed using a voter circuit that adds the partial votes of each tree in the RF and obtains the index of the class with the highest vote count [24].

Tree-based ML Datasets: The four datasets herein employed as case studies are described below:

- *AccelAdl* is a dataset for recognizing activities of daily living (ADL) using a wrist-worn accelerometer. This dataset contains 14 output classes and is available in [16].
- *Activities* is a dataset for body postures and movements classification [17]. It contains the following five classes: sitting down, standing up, standing, walking, and sitting. This dataset was collected from four different subjects.

[3]The benchmark will be available at https://github.com/Brunno815/TreeLock

[4]In the context of logic locking, an "oracle" refers to a working chip with the correct key inside. In oracle-guided attacks, the attacker has access to the oracle and can use it to obtain information about the correct key. In contrast, oracle-less attacks are conducted without access to the oracle and rely on other methods to reveal the key.

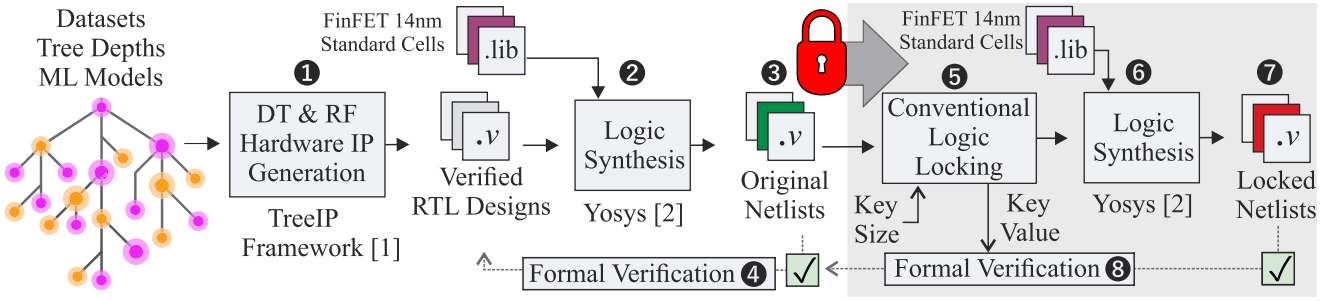

Fig. 3. The design flow for the benchmark generation. TreeIP framework [22] generates the RTLs of ML hardware IPs in step ❶ given the datasets and DT or RF depth configurations. Yosys synthesizes ❷ the HDLs and generates the original netlists ❸. An RLL script ❺ protects the netlists with a given key size. Another synthesis is performed in ❻ to obfuscate the locking gates of the locked netlists ❼. Formal verifications in steps ❹ and ❽ perform a comparison against the RTL to ensure an error-free operation of the original netlist as well as the locked netlist using the right key.

- *Wearable* is a dataset denoted as *Weight Lifting Exercises Monitored with Inertial Measurement Units*, containing five classes. Further details can be found in [18].
- *Wireless* is a dataset that focuses on activity recognition with healthy older people using a batteryless sensor. The set contains four output classes and is presented in [19].

Scikit-Learn Training Framework: RF and DT algorithm models can be trained and implemented in software using the Scikit-Learn (SKLearn) Python framework [25]. SKLearn employs classification and regression trees (CART) to build tree-based algorithms. The default information gain metric is the Gini impurity. SKLearn also offers the option of limiting the maximum tree depth for generating compact DT and RF models. Each tree in the RF is generated by using a random subset of the total number of features so that they can differ from each other. SKLearn can straightforwardly implement a software model for DTs and RFs. However, other specific generation frameworks are required to generate hardware. Therefore, the next section details how we generate the tree-based hardware IP benchmark employed in this work for investigating the effectiveness and security of logic locking.

III. Our Tree-Based Machine Learning Hardware IP Benchmark Generation

Our benchmark circuits are built with the design flow presented in Fig. 3. Hardware IPs equivalent to the SKLearn software structures can be implemented using comparisons with thresholds (i.e., employing less or equal comparators), and a combination of AND and OR gates to merge the paths of the tree to each leaf node. A framework, named TreeIP [22], is used in this work to generate the hardware IPs for the DTs and RFs. TreeIP generates hardware IPs that are able to process the features of the dataset and deliver the classification in one-hot encoding as output (i.e., with one bit per class).

The DTs in the TreeIP framework are employed using native SKLearn structures. RFs are structurally composed of the same elements as in the DT but with an additional voter module included. For RFs, a customized implementation of the voter module is employed. This voter adds the partial votes of each DT (all of which have their outputs in one-hot encoding), obtaining a final vote count for each class (more details in [24]) Then, a tree of comparators is used to ensure that the output

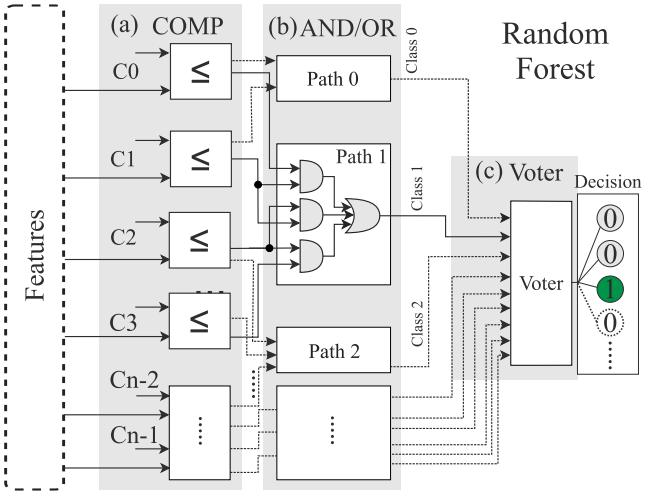

Fig. 4. Random Forest architecture divided by its sub-modules.

with the highest vote count (or lowest index, in case of a draw) is propagated to the final output in a one-hot encoding format. This slightly differs from the SKLearn voter implementation, which has different weights for the DTs.

The corresponding model of the RFs used in this work is presented in Fig. 4, considering a generic dataset. The RFs are divided into three sub-modules, which were described above: the comparators, the AND/OR paths, and the voter. The outputs considered in this work are one-hot encoded. Therefore, only one of the outputs is true in the inference, and its index defines the predicted class. Structurally, DTs contain the same elements of the RF, except for the voter sub-module which is not required due to the presence of only one tree.

To build the benchmark suite, first, the open-source TreeIP framework [24] is employed to generate the register-transfer level (RTL) designs of the DTs and RFs in step ❶. The framework trains software structures of the trees based on the datasets provided and on the stop condition (in this case, the maximum tree depth). With this software structure, it is possible to obtain accuracy using part of the dataset. TreeIP also verifies the hardware IP generated performing its equivalence with the software model in terms of accuracy.

In step ❷, we synthesize and map the RTLs generated in ❶ using Yosys Logic Synthesis Suite. We generate mapped

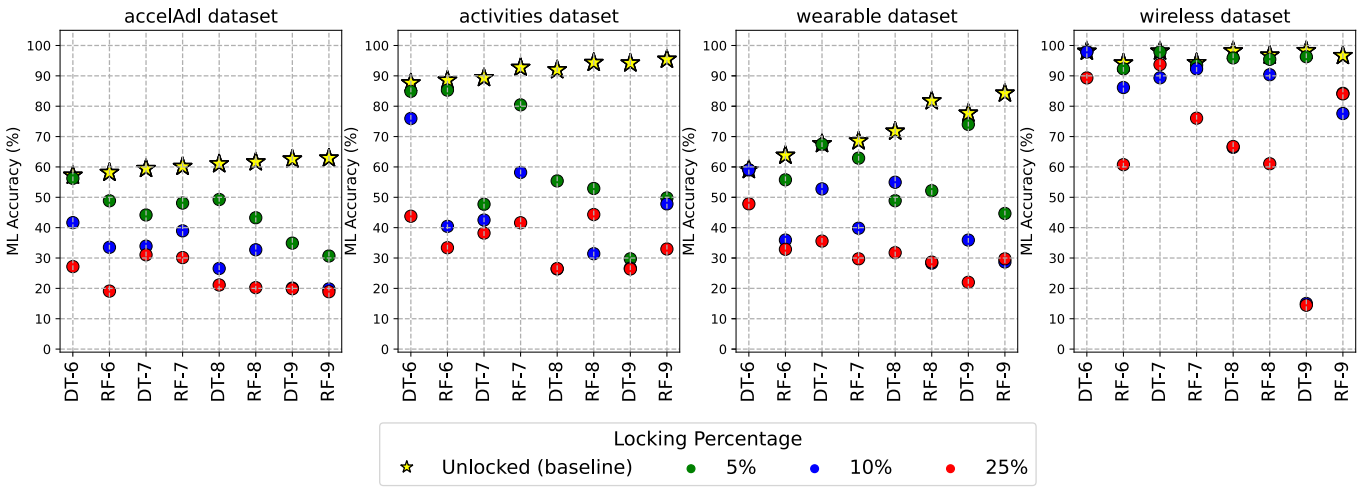

Fig. 5. Logic locking effectiveness analysis in terms of ML accuracy. The DTs and RFs, ML hardware IPs are simulated 100 times to obtain enough statistical information, each one with a different pseudo-random key. DTs and RFs are evaluated in four datasets (a) accelAdl, (b) activities, (c) wearable, and (d) wireless and four-depth configurations (from 6 to 9) with locking proportional to the respective gate count in three different percentages (5%, 10%, and 25%). ML hardware accuracy [%] – Maximum achieved with 100 simulations.

netlists ❸ using FinFET 14nm standard cell libraries fully calibrated with industrial measurements from Intel.

Then, the gate-level netlists pass through a formal verification employing Yosys to check their equivalence against their respective RTLs in step ❹. We locked the netlists ❸ using RLL in step ❺.[5] Our implementation of RLL takes in the gate-level netlist, the type of key-gate to insert (e.g., X(N)OR or multiplexer), the .lib file (for parsing purposes), which contains attributes of the standard-cell elements and the operating conditions, and the desired key size.

The locked netlists obtained from our RLL implementation contain the same gates as the original designs, along with the newly added key-gates and key-inputs. These locked netlists are then synthesized once again using Yosys ❻ to further optimize the locking structures and eliminate structural leakage. This synthesis leads to newly locked netlists ❼, which must be verified to be logically equivalent to the regular ones, as long as the correct keys are used ❽.

For this benchmark generation, we consider DTs and RFs designs from four datasets (i.e., accelAdl, activities, wearable, and wireless), with four tree depths (6, 7, 8, and 9), resulting in a total of 32 netlists. This work considers RFs composed of five trees. Furthermore, we lock the netlists with a number of keys that generate a percentage of the area overhead of the total number of gates of that respective netlist. In this work, we consider the percentage overhead values of 5%, 10%, and 25%, leading to a total of 96 locked netlists.

IV. Investigating the Effectiveness of Logic Locking in Machine Learning

Tree-based ML hardware is error-resilient due to its inherent probabilistic characteristic [14]. One of our contributions is an analysis regarding the effectiveness of logic locking considering tree-based IP hardware. This analysis considers all the benchmark circuits generated in this work.

[5]The key-gates are placed randomly in the gate-level netlist.

The effectiveness of RLL considering the tree-based models is verified by generating several pseudo-random keys and applying them to the locked netlist, analyzing the obtained ML accuracy.[6] For this investigation, we consider 100 different uniform pseudo-random keys for each of the 96 locked circuits, leading to 9600 simulations being performed.

We present in Fig. 5 the maximum classification accuracy obtained in the random analysis for all tested tree configurations, considering the three different locking percentages.

The effectiveness analysis revealed that the locked ML IPs frequently present unexpected invalid classifications (i.e., null or multiple simultaneous classifications in the single-label ML classifier). To calculate the accuracy, we consider that the attacker can make the following simple assumption in the presence of invalid classifications: in the one-hot encoded output, they can choose the class with the lowest index out of the classes that were set to logic one. When the corruption leads to no classes being inferred (i.e., null classifications), we choose the first index. The accuracy for the unlocked netlist (i.e., free from locking errors) is also presented in Fig. 5 (☆).

As it can be seen from the analysis, several points, especially when considering 5% and 10% of locking percentages, still manage to obtain acceptable accuracy results when compared to the baseline accuracy, therefore implying that more severe locking percentages are required to really stop ML functionality. Evidently, applying a locking percentage of 25% (leading to a high area overhead) seems to almost eliminate the cases with acceptable accuracy values, except for some simulations in the Wireless data set, and some in DTs with smaller depths (7 and 8).

The tendency of the circuits to still maintain high accuracy in some scenarios with small depths is also due to the fact that these circuits do not present a large area. Hence, applying locking percentages of 5% or 10% does not lead to such a large

[6]ML accuracy indicates the percentage of correct ML inferences over the total.

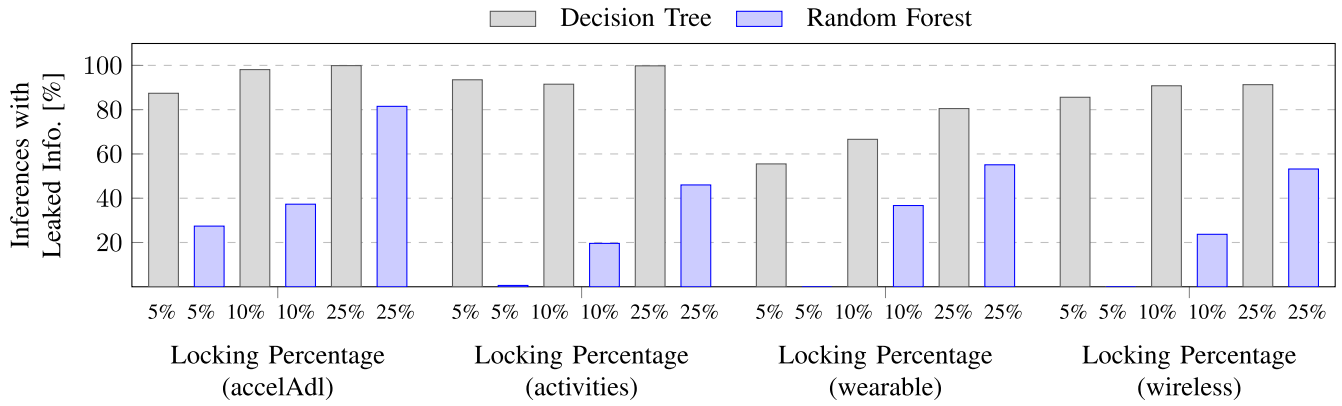

Fig. 6. Percentage of multiple-label inferences that leaks locking information, i.e., vulnerable. A vulnerable output occurs in case of multiple or null classifications.

key to many smaller DTs, making the circuit not corrupt under some wrong key combinations. Even though RFs under 5% of the locking percentage also do not present high corruption, they are mostly totally corrupted when applying 10% or more. Additionally, the larger the circuit is, the higher the number of keys that will be inserted. Thus, the probability of a key being inserted in portions of the tree-based model that may corrupt the circuit is higher as well. Overall, we can observe that corruption is not certain when employing tree-based ML models. Thus, one must carefully choose locking percentages that are high enough to make sure that the circuit does not keep working under several wrong key combinations. However, RLL may not always be the most suitable option to lock ML circuits, and new key-gate insertion mechanisms are needed to increase corruption while limiting the overhead. Note in Fig. 6 that DTs have higher vulnerability than RF because almost all inferences produce leaked information (80-100%). Although RFs apparently are more sensitive to the locking percentage (Fig. 6). It is because RFs intrinsically mask part of the leaked information in their voting mechanism, which fails to protect 100% because the voters also have errors induced by locking gates. In Section V, we demonstrate two novel solutions to effectively remove the vulnerabilities observed in unprotected circuits (Fig. 6).

V. Security Analysis of Logic Locking for Tree-Based Machine Learning

The security analysis is divided into two subsections. In the first subsection (Section V-A), a vulnerability discovered in locked ML classifiers is presented. Two countermeasure logic locking techniques are proposed and implemented to address the vulnerability, with hardware results demonstrated. In the second subsection (Section V-B), the security of locked tree-based ML circuits against attacks aimed at revealing the key is further examined. Both oracle-less and oracle-guided attack types are covered, and the countermeasure proposals are also investigated against these attacks. All analyses presented here regarding the gate area include both the ML and logic locking modules. Logic locking adds extra logic and tends to increase energy and the delay of the circuits. However, our work focuses more on demonstrating the vulnerability of the logic locking mechanism and proposing two novel, effective and low-area overhead techniques to solve it.

A. Vulnerability Identification and Proposed Countermeasures

In the effectiveness analysis presented in Section IV, we found that logic locking can cause frequent unexpected invalid inferences, where a locked ML classifier produces multiple or null classifications when tested with a wrong key. This behavior may be incorrectly interpreted as a successful logic-locking corruption that fully deteriorates the circuit output. However, for single-label ML classifiers, multiple or null inference outputs are suspicious results that signal how far the tested key is from the correct one. Thus, this unexpected invalid classification introduces a critical vulnerability that attackers can exploit to drive their attacks and attempt to find the right key. Specifically, the attackers can use the number of invalid output bits as an objective function and minimize it to evaluate if the bit-key is going in the correct direction or not at each inference, given this vulnerable feedback.

To investigate the severity of this issue, we investigate how often this logic-locking information leaking occurs. Fig. 6 shows the frequency of the invalid inferences that present leaked information (i.e., have zero or more than one classification in the one-hot encoded output). As can be seen, in general, the increase in the locking percentage tends to increase the number of vulnerable inferences. This is expected given that the higher the locking percentage is, the larger the bit width of the key is, leading to a higher chance that a locking gate was inserted in a critical portion and was set with a wrong key bit. The vulnerability in general is also less critical for RFs, given that the voter circuitry may partially correct any corruption that happens in each of their partial trees. Even though the partial trees can be corrupted, the voter module will still count the votes of the tree as long as it is not corrupted. In the case of DTs, there is no such protection, as any change in the paths of a DT may lead to an invalid output. On average for the four datasets, the DTs present percentages with leaked information of 80.5%, 86.8%, and 92.9% respectively for the locking percentages of 5%, 10%, and 25%. For the RFs, the

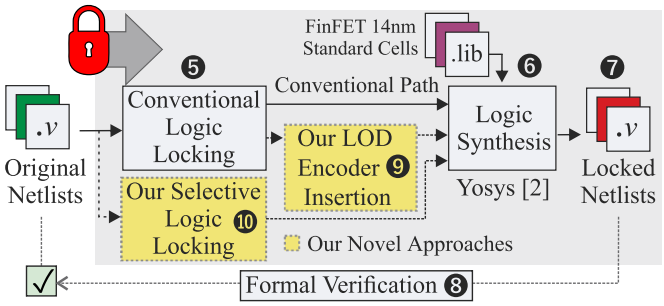

Fig. 7. Our design flow correction for solving the vulnerabilities for ML classifiers. After conventional logic locking ❺ is applied, we insert a LOD encoder ❾ to protect the circuit from vulnerabilities and perform logic synthesis ❻, leading to a newly obtained locked netlist ❼. Alternatively, our second proposal applies selective logic locking ❿ in an ungrouped version of the original netlist. The same later steps are applied.

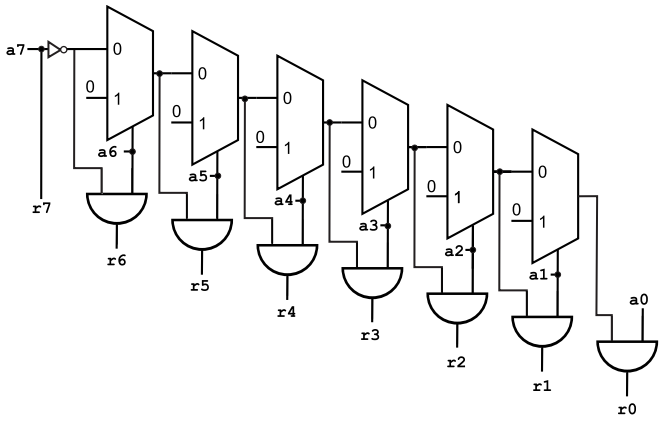

Fig. 8. An 8-bit LOD circuit example [26]. The modification consists of connecting the input a of the LOD to the output of the ML classifier being the output r of the LOD, which is the new output of the entire system.

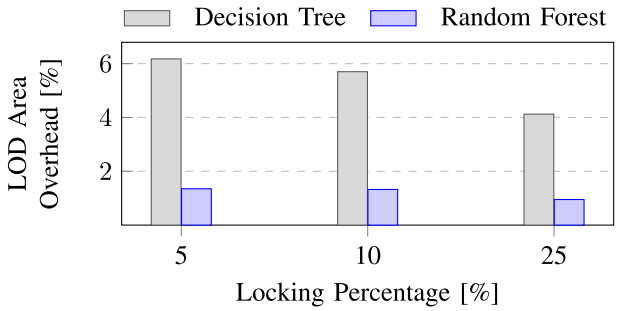

Fig. 9. Vulnerability protection costs in terms of circuit area overhead. The small area cost of the generic approach proposal (Fig. 7 - step ❾) to overcome the logic locking vulnerability issues revealed in this paper. The additional cost if for including the LOD encoder to the locked netlists of DTs and RFs ML IP classifiers for three locking percentages proportional to the circuit gate count (5%, 10% and 25%),.

average leaking percentages are 7%, 29.3%, and 59% for the same respective locking percentages.

To mitigate this issue, we propose two methods, (i) a generic one for any kind of single-label ML classifier embracing DTs and RFs, and (ii) a specific approach based on domain knowledge, which is applied for RFs. In both proposals, for all configurations, the number of total corrupted inferences drops down to zero. The proposed approaches are presented in the design flow of Fig. 7, extending the previous one from Fig. 3.

1) Our Generic Approach: inserts an additional layer of protection to the output of the ML classifier. The additional layer is a LOD encoder ❾ for choosing just one of the outputs as the correct one, ensuring a one-hot encoding output. Fig. 8 presents an 8-bit LOD circuit example [26] that can be applied to an ML classifier with eight classes. In this case, we apply the LOD after the locking, so we ensure that this portion of the circuit will not present any errors regardless of the key used by the attacker. It is important to mention that a small modification has to be done in the LOD, given that, besides having more than one output with a logic one, it is possible that the wrong key in the locked circuit may have caused every output to be logic zero. This technique can be applied to circuits other than tree-based models, as long as they involve one-hot encoding. Any circuit with one-hot encoding can be corrupted by logic locking, so using a LOD after locking the circuit keeps it protected. Additionally, we evaluate the cost in the circuit area of the LOD encoder extra layer proposal.

Fig. 9 presents the circuit area overhead of using LOD in every locked netlist, for every locking percentage. The tree depths and the datasets were averaged in this analysis.

The baseline area is the respective locked and synthesized DT/RF configuration, without LOD. As can be seen, the area increases are always below 10% of the baseline locked circuit. This is obviously more costly to the DTs, given that they present a smaller area, as RFs are composed of several DTs and are therefore larger. Furthermore, the increase in the locking percentage also tends to decrease the area overhead of inserting the LOD. This is due to the fact that applying higher locking percentages tends to increase the circuit area. Thus, the LOD will represent a smaller share of the total circuit area.

2) Our Domain-Specific Approach: exploits intrinsic voting schemes present in some classes of democratic ML models that can be preserved from logic locking to overcome invalid inferences. This approach can be applied to the RFs by selectively locking the circuit preserving its voters as presented in ❿ of the flow in Fig. 7. Our premise for this proposal realization in RFs is that the voter circuitry can be protected from the locking to prohibit multiple classifications. In other words, any error that influences the single votes of the DTs that compose an RF will be corrected (in terms of maintaining the one-hot encoding scheme valid) by the voter module. As described in Fig. 4, RFs comprise the comparators, AND/OR paths, and the voter circuitry. Therefore, the RTL descriptions have three main modules described. The proposal consists of performing an ungrouped synthesis (i.e., without applying the flattening option during the synthesis process). This synthesis generates a netlist that is separated by each module as in the RTL description. In this case, we separate both the comparators and AND/OR modules and apply the same amount of locking that would be applied in the regular netlist synthesized in a grouped manner. The number of gates applied as keys to each of the two modules will be proportional to their areas regarding each other. Hence, if the comparators module has 10 times more area than the AND/OR module, it will get 10 times more gates to be inserted as keys, as long as the sum between both is the number of keys that would be

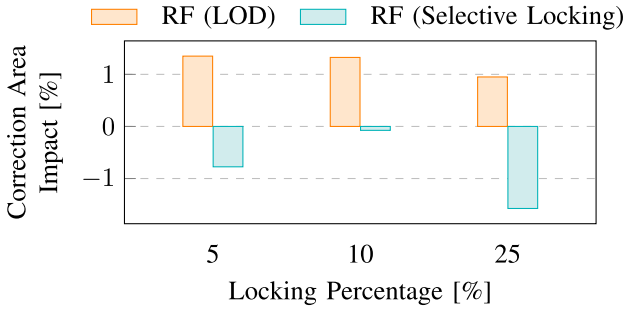

Fig. 10. Comparing the costs of the proposals for eliminating the vulnerability of RF classifiers: generic approach using LOD encoder proposal (Fig. 7 - step ❾) and the domain-specific approach through the selective logic locking proposal (Fig. 7 - step ❿). The baseline for comparison is the locked-synthesized circuit without protection approaches (i.e., vulnerable) (Fig. 3 - step ❼).

in the regular netlist. This domain-specific technique is slightly more restrictive in terms of being able to be applied to other applications, as it requires a circuit with a voter in the output, which is the case of an RF.

The results from Fig. 10 show that there is no additional cost to implement the protection proposal using our selective domain-specific approach compared to the locked circuit after synthesis as a baseline, presenting also a slight decrease in area. This analysis only contains RFs given that the proposal is specifically for them. The plot also presents the previous proposal of the LOD for RFs for comparison purposes. As it can be seen, since there is no additional circuitry involved in the selective approach, the gate area, on average for the tree depths and data sets, presents decreases ranging from 0.07% to 1.57%. This occurs given that synthesizing the circuit with ungrouped modules may cause small deviations in the overall choices of gates that the synthesis tool opts to map. After applying the selective locking in the ungrouped netlist, it is synthesized once again to obtain a grouped final netlist, which can result in a small area deviation from the originally locked circuit, leading to the small decreases observed. This proposal shows that domain knowledge is an important path to propose solutions for efficient locking, as the circuit area results are smaller than the ones using LOD for all cases (Fig. 10). The reduced area impacts when compared to LOD (and when compared to regular locking) is explained by the fact that the selective approach does not directly involve the inclusion of extra gates. It is merely a technique to apply the same logic locking to specific modules of the circuit so that the voter portion is not locked and does not corrupt the circuit.

B. Security Against Attacks

We perform security analysis in this work by applying two attacks, an oracle-less and an oracle-guided. For the oracle-less attack, we apply OMLA [27], whereas, for the oracle-guided attack, NEOS is employed [28].

OMLA is a state-of-the-art oracle-less attack that aims to unveil logic locking [27]. Instead of relying on a working chip with the key inside, OMLA utilizes a graph neural network (GNN) to map the problem of finding the correct key to a subgraph classification task. Specifically, OMLA performs subgraph extraction for each key gate in the locked netlist and uses the extracted graphs to learn the characteristics of the bit values of these gates [29].

NEOS is an open-source framework for obfuscation and deobfuscation of netlists [28]. The framework employs SAT solvers for deobfuscation, utilizing several key-condition-crunching techniques. NEOS specifically employs the well-known Glucose SAT solver, which is based on a scoring scheme for the clause learning mechanism of modern SAT solvers.

We start this sub-section by presenting the key-prediction accuracy for both oracle-less (OMLA) and oracle-guided (NEOS) approaches. This analysis is presented in Fig. 11, for the conventionally locked circuits, as well as the locked circuits with our two proposals (i.e., LOD and selective locking). This analysis is averaged for the four datasets.

As can be seen, the results are presented in different colors for the different locking percentages. The baseline is represented by a ☆ and denotes the accuracy of the correct key, which is always 100%.

Some of the attack runs using OMLA do not present an accuracy as high as the ones using NEOS, given that the oracle-guided attacks always have access to the original circuits as well. This gives the attack more information to unveil the key. This was generally true for the DTs, which led to NEOS obtaining more than 90% of the entire key in every DT. In the case of RFs, both OMLA and NEOS are not able to significantly unveil the key. This may be attributed to the only difference between DTs and RFs, which is the voter circuitry, that may be breaking the regularity of the architecture from an attacking perspective.

It is also important to notice that our proposals, in general, did not ease the attacks to unveil the key when compared to the regular locked circuit. In OMLA, when applying LOD, some configurations achieved slightly higher key-prediction accuracy, but many decreases were observed as well, especially for a locking percentage of 5% (in DT-7 and RF-9, for instance). When applying the selective approach, we significantly increased the protection of the circuit, given that OMLA presented lower key-prediction accuracy. This is due to the fact that the selective locking maintains only the comparators and the AND/OR portions of the RF locked, which are much more regular and therefore harder to attack. The NEOS oracle-guided attack did not present any difference in the key-prediction accuracy for our proposals.

The key prediction accuracy originally found when attacking non-error-resilient circuits with OMLA averaged 89.55% [27] on average for the ISCAS-85 benchmark. In our OMLA attacks with DTs and RFs circuits, we found averages of 67.8%, 68.6%, and 58.2%, respectively for the versions using locking, locking with LOD, and locking with the selective approach. Thus, we observe that while tree-based ML circuits have an intrinsic error resiliency, they are also harder to attack using structural methods given the higher regularity of their datapaths. Also, as mentioned, attacking these circuits after protecting them using the selective locking approach is even harder given that the regularity is increased in the locked portions when selectively locking only the comparators

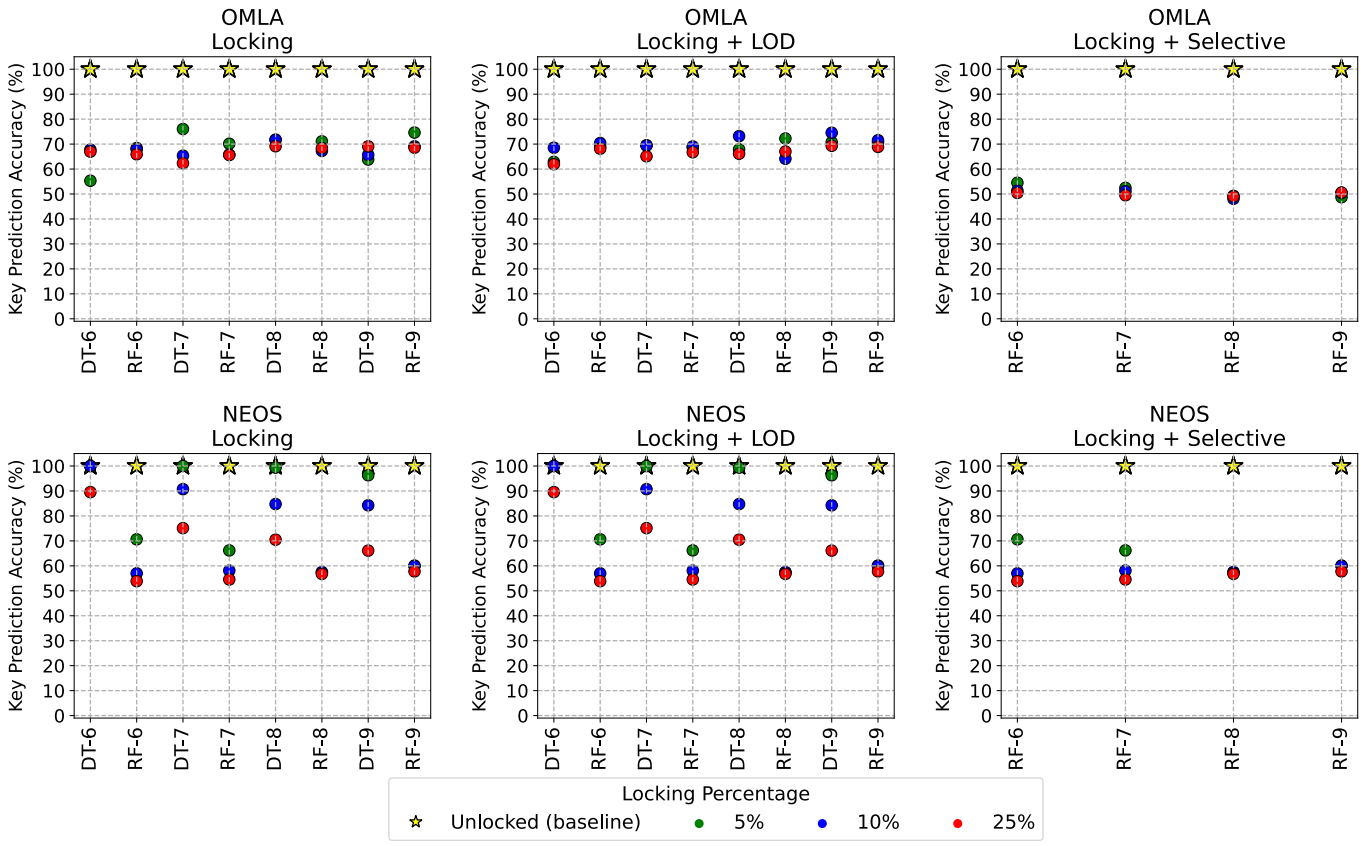

Fig. 11. Key accuracy results found by the different attack methods OMLA, and NEOS, for the different tree depths, locking percentages, and for all proposed solutions. This analysis presents the average of the four aforementioned datasets.

and AND/OR modules (i.e., preventing locking in the voter path).

Overall, our proposals managed to eliminate the vulnerability identified in the one-hot encoding scheme while not incurring increases in the key-prediction accuracy for both tested attacks.

Even though it is important to evaluate how much of the key is unveiled by the attacks, the end goal takes into account the inference accuracy resulting from that unveiled key. Fig. 12 presents the inference accuracy results of the attacks using the respective predicted keys. This analysis is also averaged for the four datasets. For this analysis, the baseline results (☆) do not present 100% accuracy given that the inference accuracy highly depends on the dataset, the chosen depth, and whether the circuit is a DT or an RF.

The findings in this analysis corroborate what was found in the previous one. In general, NEOS obtained accuracy results closer to the baseline, given that it is an oracle-guided attack and thus has more information to find a more correct key. Even so, some specific accuracy results were still better for OMLA, mainly for RFs. We can still observe that the inference accuracies present the same behavior for the different techniques (normal locking and our two proposals). In very few cases, the accuracy is slightly increased when applying locking with our proposals, but the general behavior is that the inference accuracy presents similar or decreased results when compared to regular locking. This reinforces that our proposals do not leak more information to the attacker, in general.

TABLE I
SUMMARY OF THE RELATED WORKS

	Summary Results					
Related Work	A	B	C	D	E	F
[30]	✓	✓	✓	–	≈	–
[31]	✓	✓	✓	–	≈	–
[32]	✓	–	✓	–	–	–
[15]	≈	✓	–	–	≈	–
This work	✓	✓	–	✓	✓	✓

✓: Includes the respective characteristic.
≈: Partially includes.
–: Does not include.
(A) Investigated how to protect ML hardware intellectual property.
(B) Explored key-based locking techniques for ML.
(C) Locking techniques only for NNs.
(D) Found and correct the vulnerability of the locking in single-label ML classifiers.
(E) Explored the security and effectiveness of the locking techniques embracing oracle-guided and oracle-less attacks.
(F) Investigated logic locking (LL) effectiveness in tree-based ML (as DTs and RFs) hardware designs.

C. Related Work Comparisons

We performed thorough research in the literature regarding related works that employ the logic locking technique. Some works have investigated IP protection techniques in ML specific for neural network (NN) models in [15], [30], and [31]. The work in [30] proposed a domain-specific

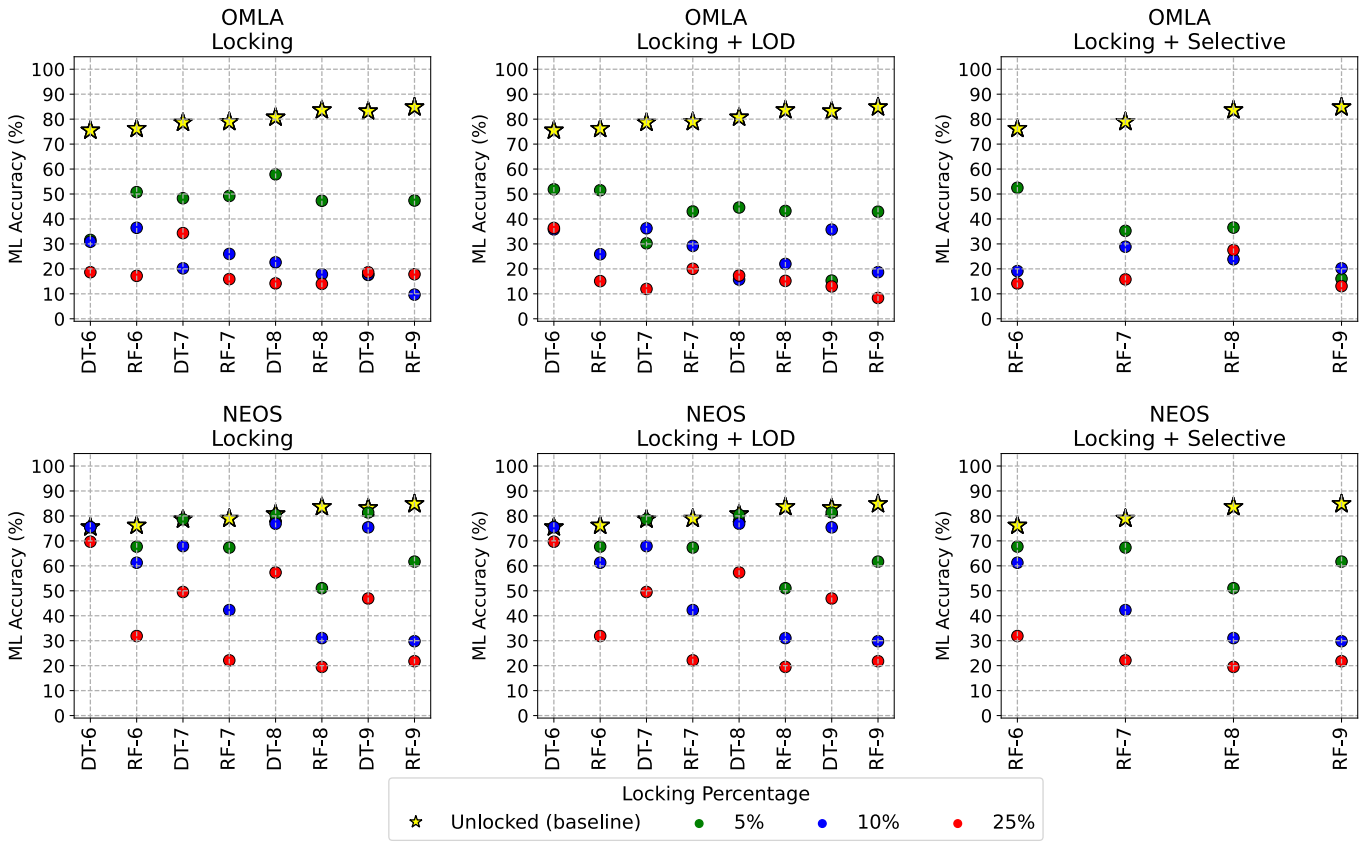

Fig. 12. Accuracy results found by the different attack methods OMLA and NEOS, for the different tree depths, locking percentages, and for the different proposed solutions. This analysis presents the average of the four aforementioned datasets.

technique to obfuscate DNN IP by training along with the secret key. Reference [31] proposes a lightweight method to protect DNNs, denoted as NN-lock. This method encrypts each parameter of a DNN model that has already been trained. This is done by employing a secret key using a scheduling mechanism. Reference [15] presents logic locking analyses for approximate adders and multipliers, as well as full NN accelerators. The results present the low corruption of these solutions depending on the locking techniques employed. This work, for the first time, evaluates logic locking for tree-based ML models. Moreover, the existing works do not perform an analysis using *both* oracle-less and oracle-guided attacks and do not address any vulnerabilities related to the one-hot encoding scheme. Due to the lack of related works directly associated with logic locking and tree-based ML models from our literature research, a quantitative comparison is infeasible to be performed. Even so, a summary of the characteristics of the main related works found is presented in Table I.

VI. Conclusion

In this work, we addressed the challenges associated with logic locking in the context of machine learning (ML) hardware. Hereby, we present, for the first time, a deep investigation of logic locking for tree-based ML hardware. First, we generate a full benchmark suite of locked gate-level decision trees (DTs) and random forests (RFs) netlists, containing 96 different circuits. Second, we investigate the effectiveness of the logic locking in terms of ML accuracy drop considering random keys. The analysis showed that we can still obtain high accuracy with wrong keys, depending on the locking percentages used (i.e., mainly in 5% and 10%). Further, we identify a critical vulnerability in the logic-locked tree-based models, i.e., providing invalid outputs for wrong keys, allowing for rapid key pruning. This vulnerability represents an information leakage, undermining the security of logic locking. We address the vulnerability using two approaches: (i) The use of a lead-one detector (LOD) circuit in the output of the locked netlist to correct the invalid outputs for both DTs and RFs. (ii) The use of constrained logic locking to avoid corrupting the voter circuitry, for RFs specifically. Both these proposals completely eliminate the number of invalid inferences, overcoming the vulnerability, for a small cost in the circuit area (and in some cases slightly decreasing, for the selective locking technique). Then, we perform a thorough security analysis in these locked netlists, by applying oracle-less (OMLA) and oracle-guided (NEOS) attacks known from the literature, to find the amount of correct key bits unveiled by each of these. Finally, we investigate the effectiveness of the logic locking in terms of inference accuracy using the unveiled keys from both attacks. We found that oracle-less attacks are less effective in our DT/RF accelerators when compared to exact circuits analyzed in previous works. With our provided investigation and locking solutions, we establish the cornerstones for the protection of ML hardware using logic

locking. Future research can explore similar vulnerabilities in other low-cost ML circuits, such as hyper-dimensional computing (HDC).

The key contribution of this paper resides at the logic level of abstraction, where we identify and address a logic insecurity problem that is independent of the electrical characteristics of the underlying technology. The unveiled insecurity will appear in any unprotected ML classifiers that use Logic Locking implemented using technologies based on the Boolean theory, which embraces all the current commercial technologies (e.g. any FPGA, CMOS, etc.). Further studies still can be made for emerging and unconventional technologies.

ACKNOWLEDGMENT

The authors would like to thank Tim Bücher for his support.

REFERENCES

[1] N. Kühl, M. Goutier, L. Baier, C. Wolff, and D. Martin, "Human vs. supervised machine learning: Who learns patterns faster?" *Cognit. Syst. Res.*, vol. 76, pp. 78–92, Dec. 2022.

[2] D. Schoder, "Introduction to the Internet of Things," in *Internet of Things A to Z*, Q. Hassan, Ed., Hoboken, NJ, USA: Wiley, 2018, doi: 10.1002/9781119456735.ch1.

[3] J. Portilla, G. Mujica, J.-S. Lee, and T. Riesgo, "The extreme edge at the bottom of the Internet of Things: A review," *IEEE Sensors J.*, vol. 19, no. 9, pp. 3179–3190, May 2019.

[4] K. Balaskas, G. Zervakis, K. Siozios, M. B. Tahoori, and J. Henkel, "Approximate decision trees for machine learning classification on tiny printed circuits," in *Proc. 23rd Int. Symp. Quality Electron. Design (ISQED)*, 2022, pp. 1–6.

[5] C. Xu, Y. Song, M. Han, and H. Zhang, "Portable and wearable self-powered systems based on emerging energy harvesting technology," *Microsyst. Nanoeng.*, vol. 7, no. 1, pp. 1–14, Mar. 2021.

[6] E. Tabanelli, G. Tagliavini, and L. Benini, "Optimizing random forest-based inference on RISC-V MCUs at the extreme edge," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 11, pp. 4516–4526, Nov. 2022.

[7] G. Pedretti et al., "Tree-based machine learning performed in-memory with memristive analog CAM," *Nature Commun.*, vol. 12, no. 1, pp. 1–10, Oct. 2021.

[8] J. Dean, D. Patterson, and C. Young, "A new golden age in computer architecture: Empowering the machine-learning revolution," *IEEE Micro*, vol. 38, no. 2, pp. 21–29, Mar. 2018.

[9] W. J. Dally, Y. Turakhia, and S. Han, "Domain-specific hardware accelerators," *Commun. ACM*, vol. 63, no. 7, pp. 48–57, Jun. 2020, doi: 10.1145/3361682.

[10] D. Sisejkovic, F. Merchant, L. M. Reimann, and R. Leupers, "Deceptive logic locking for hardware integrity protection against machine learning attacks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 6, pp. 1716–1729, Jun. 2022.

[11] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending piracy of integrated circuits," in *Proc. Conf. Des. Automat. Test Europe*, 2008, pp. 1069–1074.

[12] B. Tan et al., "Benchmarking at the frontier of hardware security: Lessons from logic locking," 2020, *arXiv:2006.06806*.

[13] S. Zhao, N. Shah, W. Meert, and M. Verhelst, "Discrete samplers for approximate inference in probabilistic machine learning," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 1221–1226.

[14] B. A. de Abreu, G. Paim, M. Grellert, and S. Bampi, "C2PAx: Complexity-aware constant parameter approximation for energy-efficient tree-based machine learning accelerators," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 7, pp. 2683–2693, Jul. 2022.

[15] K. Nayak, D. Upadhyaya, F. Regazzoni, and I. Polian, "On the limitations of logic locking the approximate circuits," in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, Dec. 2022, pp. 1–6.

[16] D. Dua and C. Graff. (2017). *UCI Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml

[17] W. Ugulino, D. Cardador, K. Vega, E. Velloso, R. Milidiu, and H. Fuks, "Wearable computing: Classification of body postures and movements (PUC-Rio)," UCI Mach. Learn. Repository, 2012, doi: 10.24432/C54W37.

[18] E. Velloso, A. Bulling, H. Gellersen, W. Ugulino, and H. Fuks, "Qualitative activity recognition of weight lifting exercises," in *Proc. 4th Augmented Hum. Int. Conf.*, New York, NY, USA, Mar. 2013, pp. 116–123.

[19] T. Shinmoto et. al., "Sensor enabled wearable RFID technology for mitigating the risk of falls near beds," in *Proc. IEEE Int. Conf. RFID (RFID)*, Jun. 2013, pp. 191–198.

[20] M. Martins et al., "Open cell library in 15 nm FreePDK technology," in *Proc. Int. Symp. Phys. Design (ISPD)*, 2015, pp. 171–178.

[21] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. Int. Symp. Hardw. Orient. Secur. Trust (HOST)*, 2015, pp. 137–143.

[22] (2020). *TreeIP: Framework for Tree-based Hardware IP Generation*. [Online]. Available: https://github.com/Brunno815/Framework_DTRF

[23] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 3133–3181, Jan. 2014.

[24] B. Abreu, M. Grellert, and S. Bampi, "A framework for designing power-efficient inference accelerators in tree-based learning applications," *Eng. Appl. Artif. Intell.*, vol. 109, Mar. 2022, Art. no. 104638.

[25] F. Pedregosa, S. Varoquaux, A. Gramfort, V. Michel, and B. Thirion, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Dec. 2011.

[26] K. H. Abed and R. E. Siferd, "CMOS VLSI implementation of a low-power logarithmic converte," *IEEE Trans. Comput.*, vol. 52, no. 11, pp. 1421–1433, Nov. 2003.

[27] L. Alrahis, S. Patnaik, M. Shafique, and O. Sinanoglu, "OMLA: An oracle-less machine learning-based attack on logic locking," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 3, pp. 1602–1606, Mar. 2022.

[28] K. Shamsi and Y. Jin. (2019). *NEOS: Netlist Encryption and Obfuscation Suite*. [Online]. Available: https://cadforassurance.org/tools/evaluation-of-obfuscation/neos/

[29] L. Alrahis et al. (2021). *OMLA: An Oracle-Less Machine Learning-Based Attack on Logic Locking*. [Online]. Available: https://github.com/DfX-NYUAD/OMLA

[30] A. Chakraborty, A. Mondai, and A. Srivastava, "Hardware-assisted intellectual property protection of deep learning models," in *Proc. 57th ACM/IEEE Des. Automat. Conf.*, Jul. 2020, pp. 1–6.

[31] M. Alam, S. Saha, D. Mukhopadhyay, and S. Kundu, "NN-lock: A lightweight authorization to prevent IP threats of deep learning models," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 18, no. 3, pp. 1–19, Apr. 2022, doi: 10.1145/3505634.

[32] M. Xue, J. Wang, and W. Liu, "DNN intellectual property protection: Taxonomy, attacks and evaluations (invited paper)," in *Proc. Great Lakes Symp. VLSI*. New York, NY, USA: Association for Computing Machinery, Jun. 2021, pp. 455–460, doi: 10.1145/3453688.3461752.

Brunno Alves de Abreu (Student Member, IEEE) received the degree (Hons.) in computer engineering and the M.Sc. and Ph.D. degrees in microelectronics from the Federal University of Rio Grande do Sul (UFRGS) in 2017 and 2024, respectively. During the Ph.D. study, he was a Visiting Researcher with the Instituto de Engenharia de Sistemas e Computadores–Investigação e Desenvolvimento (INESC-ID), Lisbon, Portugal, contributing to his doctoral research. Currently, he is a Senior Engineer developing standard cell characterization with Silvaco Company, Porto Alegre, Brazil. He received the Best Student Award from the Brazilian Computing Society.

Guilherme Paim (Member, IEEE) received the bachelor's degree (Hons.) in electronics engineering from the Universidade Federal de Pelotas (UFPel), Brazil, in 2015, and the Ph.D. degree (summa cum laude) in microelectronics from the Universidade Federal do Rio Grande do Sul (UFRGS), Brazil, in 2021. He is an Assistant Professor with the Instituto Superior Técnico (IST), University of Lisbon, Portugal, and an Integrated Researcher with INESC-ID, Lisbon. During the Ph.D. study, he conducted part of his research with Karlsruhe Institute of Technology (KIT) in collaboration with the University of Stuttgart, Germany (20192020). His research experience includes working with MICAS Laboratories, KU Leuven, Belgium, where he focused on RISC-V multi-core artificial intelligence system-on-chips. He also collaborates closely with the System-Technology Co-Optimization (STCO) team with imec R&D, Begium. With around 100 research papers published in conferences and journals, he is a significant contributor to the field of chip design and its software stack. He serves as a technology transfer consultant for microelectronics projects for the Brazilian Ministry of Science, Technology, and Innovation (MCTI). He has co-authored the Brazilian Plan for AI (PBIA). He received the Best Ph.D. Thesis Award from the Brazilian Microelectronics Society (SBMicro) and the Honor Thesis Award from the CAPES Agency in 2022. Additionally, he is a member of the Technical Program Committee (TPC) of several conferences, including DATE, ASP-DAC, TinyML, MLCAD, VLSI-SoC, and IEEE LASCAS. His website: gppaim.wordpress.com.

Lilas Alrahis (Member, IEEE) received Ph.D. degree in electrical and computer engineering from Khalifa University, United Arab Emirates, in 2021. Since then, she has been a Post-Doctoral Associate with the Engineering Division, New York University Abu Dhabi (NYUAD), United Arab Emirates. In Fall 2024, she is assigned as an Assistant Professor with the Computer and Information Engineering Department, Khalifa University. Her current research interests include hardware security, design-for-trust, and applied machine learning. She received the MWSCAS Myril B. Reed Best Paper Award in 2016 and the Best Paper Award at the Applied Research Competition held in conjunction with Cyber Security Awareness Week in 2019 and 2021. She received the Karlsruhe Institute of Technology (KIT) International Excellence Fellowship, Germany, for 2023. She is the Chair for the IEEE Women in Engineering Affinity Group, United Arab Emirates. She is serving as an Associate Editor for the *Integration, VLSI Journal*.

Paulo Flores (Senior Member, IEEE) received the Engineering, M.Sc., and Ph.D. degrees in electrical and computer engineering from the Instituto Superior Técnico (IST), Technical University of Lisbon, Portugal, in 1989, 1993, and 2001, respectively. Since 1990, he has been teaching with IST, University of Lisbon, where he is currently an Assistant Professor with the Department of Electrical and Computer Engineering. He has also been with the Instituto de Engenharia de Sistemas e Computadores (INESC-ID), Lisbon, since 1988, where he is a Senior Researcher with the High-Performance Computing Architectures and Systems (HPCAS) Group. He has contributed to more than 85 papers in journals and international conferences. His research interests are computer architecture and CAD for VLSI circuits in the area of embedded systems, testing, and verification of digital systems; and computer algorithms, with a particular emphasis on the optimization of hardware/software problems using satisfiability (SAT) models. He is a Senior Member of the IEEE Circuit and Systems Society.

Ozgur Sinanoglu (Senior Member, IEEE) received the Ph.D. degree in computer science and engineering from the University of California at San Diego.

He is a Professor of electrical and computer engineering with New York University (NYU) Abu Dhabi. He is the Director of the Center for CyberSecurity, NYU Abu Dhabi. He has an industry experience with TI, IBM, and Qualcomm. His research interests include design-for-test, design-for-security, and design-for-trust for VLSI circuits, where he has more than 200 conference papers and journal articles and 20 issued and pending U.S. patents. His recent research is being funded by the U.S. National Science Foundation, the U.S. Department of Defense, Semiconductor Research Corporation, Intel Corporation, and Mubadala Technology. During the Ph.D. study, he won the IBM Ph.D. Fellowship Award (twice). He was a recipient of the Best Paper Awards at IEEE VLSI Test Symposium 2011 and ACM Conference on Computer and Communication Security 2013.

Sergio Bampi (Senior Member, IEEE) received the Electronics Engineer and B.Sc. degrees in physics from the Universidade Federal do Rio Grande do Sul (UFRGS), Brazil, in 1979, and the M.S.E.E. and Ph.D. degrees in electrical engineering from Stanford University in 1982 and 1986, respectively. He is a Full Professor with the Informatics Institute, UFRGS, where he joined in 1981. He has published more than 500 research papers in conferences and journals in the fields of CMOS analog, digital, and RF design, video coding algorithms, and hardware architectures. He was a former President of the Brazilian Microelectronics Society and the FAPERGS Brazilian Research Funding Agency and the CEITEC Technical Director. He served as the Technical Program Chair for SBCCI in 1997 and 2005, IEEE LASCAS in 2013, VARI in 2015, SBMICRO Congress in 1989 and 1995. He served on the TPC Committees of DAC, ICCAD, SBCCI, ICM, LASCAS, VLSI-SoC, ICECS, and many other international conferences. He was a Distinguished Lecturer with the IEEE CAS Society, from 2009 to 2010.

Hussam Amrouch (Member, IEEE) received the Ph.D. degree (summa cum laude) from Karlsruhe Institute of Technology (KIT) in 2015. He is a Professor heading the Chair of AI Processor Design (AI-Pro), Technical University of Munich (TUM), Germany; and the Head of the Chair of Semiconductor Test and Reliability (STAR), University of Stuttgart, Germany. Prior to that, he was a Research Group Leader with KIT, where he was leading the research efforts in building dependable embedded systems. He has 270 publications in multidisciplinary research areas (including over 115 journals) across the entire computing stack, starting from semiconductor physics to circuit design all the way up to computer-aided design and computer architecture. His research in HW security and reliability have been funded by the German Research Foundation (DFG), Advantest Corporation, and the U.S. Office of Naval Research (ONR). His main research interests are design for reliability and testing from device physics to systems, machine learning for CAD, HW security, approximate computing, and emerging technologies, with a special focus on ferroelectric devices. He holds eight HiPEAC Paper Awards and three best paper nominations at top EDA conferences: DAC'16, DAC'17 and DATE'17 for his work on reliability. He has served on the Technical Program Committees of many major EDA conferences, such as DAC, ASP-DAC, and ICCAD. He is a reviewer of many top journals, such as *Nature Electronics*, IEEE TRANSACTIONS ON ELECTRON DEVICES, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, and IEEE TRANSACTIONS ON COMPUTERS. He serves as an Editor for *Nature Scientific Reports*.