Untangling IP Protection via Learning and Structure

Dake Chen[1*], Subhajit Dutta Chowdhury[1*], Xuan Zhou[1*], Peter A. Beerel[1,2], Pierluigi Nuzzo[1], and Matthew French[2]
[1]Ming Hsieh Department of Electrical and Computer Engineering
[2]Information Sciences Institute
Viterbi School of Engineering, University of Southern California
{dakechen, zhouxuan, duttacho, pabeerel, nuzzo}@usc.edu, mfrench@isi.edu

***Abstract*—Security is an increasingly important consideration in the context of hardware design and intellectual property protection, with defense methods spanning both combinational and sequential circuits. However, these defenses must contend with a variety of emerging attack strategies, including those based on Boolean satisfiability (SAT) solving and machine learning (ML). This paper shows, via examples, how new attacks can also emerge from the combination of data (learning) and structure (constraint solving), and reveal potential vulnerabilities that must be accounted for when constructing effective defenses. We discuss an effective oracle-less attack on latch-based logic locking that uses an ML classifier to generate the objective function coefficients of an integer linear program, capturing the structure of feasible classifications, whose optimal solutions can uncover the secret key. We then present a method that combines graph neural networks with structural analysis to classify the registers of a circuit and facilitate its reverse engineering. Our attack on latch locking is the first to be successful, finding the correct key in 8 out of 19 benchmark circuits tested and, on average, finding keys that are 98% correct. Our register classification approach significantly improves on a state-of-the-art heuristic register classification technique, yielding an average sensitivity and accuracy of over 96% on a variety of benchmark cores.**

I. Introduction

Because integrated circuit (IC) design and manufacturing rely on a global supply chain, security against intellectual property (IP) piracy, counterfeiting, and hardware Trojan insertion and the protection of IP rights have become a significant concern. Logic locking (LL), circuit redaction, and other common approaches rely on incorporating programmable logic to protect hardware IPs [1]. When the user applies an incorrect programming, or key, value to the locked circuit, the circuit outputs will be incorrect, thus locking the correct functionality. On the other hand, reverse engineering is a promising tool to restore trust in the IC supply chain by enabling the identification of malicious modifications in a design.

For many years, attacks based on Boolean satisfiability (SAT) solving have offered powerful analysis tools to benchmark IP protection methods, by relying on oracle access to an unlocked circuit. However, with the advances in machine learning (ML), protection schemes must also be designed to ensure that they do not leak structural signatures [2] that can be identified, for example, by a trained classifier [3], [4]. Furthermore, in this paper, we contend that a new breed of attacks can emerge, even in an oracle-less setting, from the combination of data, harnessed by learning, with structure, exploited by formal analysis and constraint solving methods. These new attacks can reveal potential vulnerabilities of existing protection schemes that must be accounted for when constructing effective defenses.

We illustrate this category of attacks via two examples. We first discuss an oracle-less attack to latch-based logic locking [5] that uses an ML classifier to generate the objective function coefficients of an integer linear program (ILP), capturing the structure of feasible classifications, whose optimal solutions can uncover the secret key. We then present a method that combines graph neural networks (GNNs) with structural analysis to classify the registers of a circuit and facilitate its reverse engineering while achieving high accuracy and generalization capabilities [6]. GNNs are particularly effective in processing circuit netlists in terms of graphs and leveraging properties of the nodes and their neighborhoods to learn to efficiently discriminate between different types of nodes. Structural analysis of the netlist graph further rectifies potential misclassifications.

II. ML-ILP Attack on Latch Locking: An Overview

In the following, we describe the ML-ILP attack on latch-based logic locking (LBLL), also referred to as latch locking, in which the secret key is needed to correctly configure the operation of both functional and decoy latches in the design [5].

A. Latch Locking

As illustrated in Figure 1, latch locking consists of four steps. A community detection algorithm is first used to select a subset of FFs (step 1). Each selected FF is then duplicated (step 2) and re-timed before being replaced with latches (step 3). Lastly, delay and logic decoy latches are randomly inserted into the netlist (step 4). Delay decoys, when keyed correctly, are forced to be transparent and thus only influence the delay of the circuit. Logic decoys, on the other hand, when keyed correctly, output a fixed 0 value. Their insertion must be coupled with extra OR/XOR/MUX gates to ensure the latch, when keyed correctly, does not corrupt the circuit's

∗ = Equal Contribution

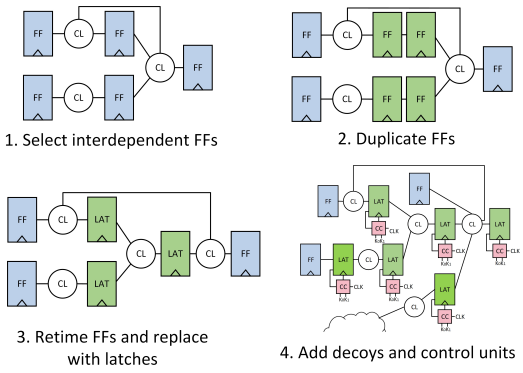

Fig. 1. The four steps of Latch-Based Logic Locking (LBLL).

functionality. Control logic is added to each latch that accepts two-bit keys to configure the four types of latches.

If we correctly classify the latches as master, slave, delay decoy, and logic decoy, we break the lock. Our attack strategy is based on the observation that the sequential graphs associated with master-slave latch-based designs have a regular structure that is broken by the random insertion of decoy latches. While ML algorithms can be trained to sometimes identify this distinction, it is more challenging for ML training approaches to incorporate the specific structure associated with a correct classification. This is particularly true if the classifier is designed to work locally, such as classifying an individual latch as a decoy, master, or slave, while the structural constraints are based on the global connectivity between latches, i.e., that the sequential graph obtained after all decoys are properly removed is two-colorable. We address this issue by combining machine learning and constrained optimization. In particular, as further detailed below, we adopt an approach as in [7], taking the soft-outputs of an ML classifier as the coefficients of the objective function of a mixed integer linear program whose constraints capture the structure of the correct solutions.

B. ML-ILP Attack

The key observation is that individually classifying latches, while powerful on its own, is more powerful if used to guide an ILP to find global solutions that also adhere to the graph coloring constraints. To do this, we use the soft-max outputs as the coefficients of the ILP objective function and add constraints that ensure that the sequential graph, once the identified decoys are removed, adheres to the coloring constraints. Moreover, the ILP can be configured to find a large pool of classifications that can be independently evaluated by the attacker.

Every latch in the circuit can be associated with a vector of structural features that can be used to train an ML classifier to identify decoy latches. Several of these features, illustrated in Figure 2, identify insertions of decoy latches that yield small sub-graphs that are not two colorable. For the ILP to capture the global constraints, each latch is associated with four T variables, T_M, T_S, and T_{DD}, and T_{LD}, denoting the latch type,

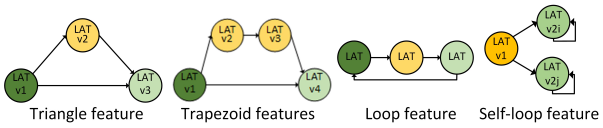

Fig. 2. Triangle, trapezoid, loop, and self-loop features

and one C variable, specifying its "color." The ILP objective is to maximize a weighted sum of the latch variables over the total number of latches, where the weights are softmax probabilities obtained from the earlier ML classifier.

We generate three groups of constraints for the ILP. The first group ensures that each latch must be classified into exactly one type of latch. The second group correlates the latch T variable to its color C such that the C variable of every master latch is equal to 1 and that of every slave latch is equal to 0. The third group of constraints help disambiguate the coloring options and avoid the assignment of master and slave to be switched. This group of constraints also ensure that a master latch cannot be followed by a master latch and a slave latch cannot be followed by a slave latch. Finally, further constraints ensure the color of decoy latches should be the same as any driving latch and the color of a master/slave latch should be different than that of any fan-in decoy, ensuring the coloring constraints pass through chains of decoy latches.

Capturing the impact of logic decoys is particularly tricky because a logic decoy emits a static '0' and their random insertion to the netlist using MUXes sometimes creates false paths between other latches. Thus, when one or more latches are classified as a logic decoy, other edges in the sequential graph can become false and should not influence coloring. This complexity can be handled by either pre-classifying logic decoys in a first pass and removing them from the circuit or by capturing the implications of logic decoys within the ILP constraints. We present evaluation results using the first approach in Section IV.

Overall, the ML-ILP attack attempts at reverse-engineering the latches of a latch-locked netlist by training a simple ML model with features that capture the structure of specific netlist sub-graphs. The outcome from the ML model is then used as a guideline to find latch classifications that satisfy more complex graph-coloring constraints encoded by an ILP. These constraints act as a form of *inductive bias* for the overall learning task [8], encoding the additional set of assumptions required to deductively infer a concept (e.g., the latch classification function) from the inputs to the learning algorithm (e.g., the labelled features).

On the other hand, in the following, we introduce another method to reverse-engineer the registers of an IC, where structural properties of graphs can be directly leveraged by the deep learning model, namely, a graph neural network. Training can then be performed using simpler features, capturing basic properties of the graph nodes and their neighborhoods.

III. Register Classification Using GNNs and Structural Analysis: An Overview

Hardware reverse engineering consists in extracting the high-level functionality of a design through multiple automated or manual analysis steps. A first step is to obtain the gate-level netlist from a fabricated chip through delayering, imaging, and post processing [9]–[11], aiming to combine the information provided from a set of images into a netlist. Once the netlist is available, gate-level netlist reverse-engineering techniques are employed to infer the circuit functionality.

In the following, we provide some background on netlist reverse-engineering and the identification of the state registers, which is often considered the most critical step in reverse-engineering the control logic of a design. We then describe a ML-based register classification technique that combines GNNs with structural analysis to classify the registers.

A. Netlist Reverse-Engineering and Register Identification

Netlist reverse-engineering consists of two main tasks, namely, data-path reverse-engineering and control-logic reverse-engineering. The control logic in a digital circuit is in charge of generating signals to control the data path. Due to their regularity and high degree of replication, data-path blocks are more suitable for algorithmic analysis, using techniques like template matching, behavioral matching of an unknown sub-circuit against a library of abstract components, or ML-based circuit recognition [12]–[14]. On the other hand, the control logic of a design is generally designed for a specific functionality, which makes it difficult to apply the techniques mentioned above.

Reverse-engineering the control logic of a design involves the identification of the state registers followed by a finite state machine (FSM) extraction step, which aims to recover the state transition graph [15], [16] of the circuit FSM. Correct identification of the state registers is, therefore, instrumental for the extraction of a correct FSM. Many techniques have been proposed to identify the state registers in a design [17]–[19]. For example, techniques like RELIC [17] and fastRELIC [18] compare the logic and the topologies of the registers' fan-in circuits to determine the type of registers. Logical and topological similarity tests are performed between all pairs of registers' fan-in circuits. Since data-path components exhibit more regularity and replication of circuit structures, including the fan-in circuits of data registers, these registers tend to achieve higher similarity scores than state registers. However, the classification accuracy can significantly rely on fine-tuning of the similarity threshold parameters for each circuit.

When little or no additional information is available about a design, it becomes challenging to converge to the parameter values that lead to the best result. Moreover, the complexity of these techniques increases polynomially with the number of registers in a design, leading to a classification time that scales worse than linearly with the circuit size.

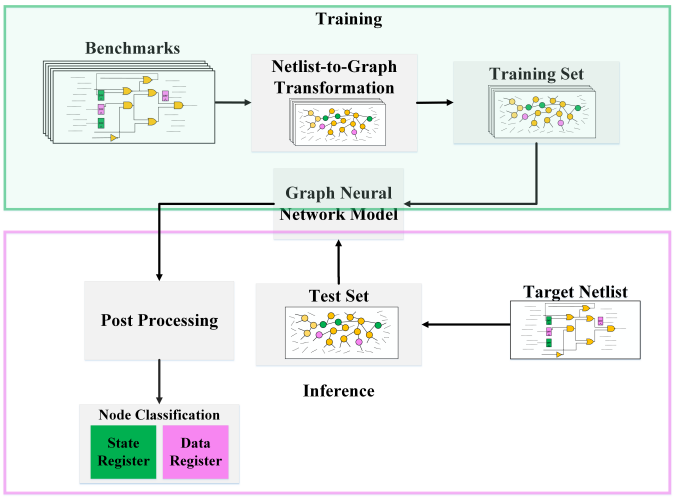

Fig. 3. ReIGNN methodology.

B. State Register Identification Using ML

Inspired by the success of state-of-the-art deep learning (DL) methods in solving different challenging problems across different fields, we have developed ReIGNN, a novel learning-based technique to classify the state and data registers in a netlist. The proposed technique combines GNNs with structural analysis to achieve high accuracy and generalization properties across different designs. GNNs are deep learning models that operate directly on graphs and adeptly perform different graph-related learning and inference tasks, like node classification, by leveraging the graph structure, the properties of the nodes, and the characteristics of their neighborhoods. By mapping the circuit gates and registers to graph nodes and their connections to graph edges, the register classification task directly translates into a node classification problem.

In the proposed technique, we automatically abstract the circuit netlist as a graph and associate each graph node with a feature vector that captures information about its functionality and connectivity. We then train the GNN that processes these graphs to discriminate between state and data registers in the netlist. The trained GNNs can then be used to classify the registers of new netlists. Finally, we perform structural analysis of the netlist and, specifically, identification of strongly connected components that include registers in the graph, to further improve the accuracy of the inference task and rectify misclassifications.

C. ReIGNN Methodology

Figure 4 illustrates the ReIGNN framework for classifying state and data registers in a netlist. We first train the GNN pipeline with a set of benchmarks and then leverage the trained GNN to perform inference on new netlists. The first step in training is to transform a circuit netlist into a graph and associate the graph nodes with feature vectors and labels to create the training set. The feature vector of a node contains the following information: (a) node type, (b) node in-degree, (c) node out-degree, (d) node betweenness centrality, (e) node harmonic centrality, and (f) number of different types of gates

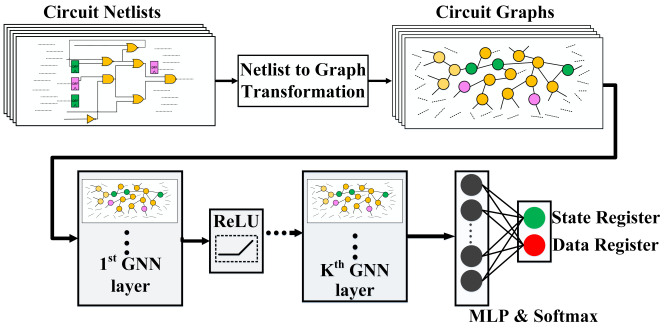

Fig. 4. GNN model architecture.

appearing in the neighborhood of the node. We use one-hot encoding to represent the node type. Intuitively, betweenness centrality is an indicator of the influence of a node over the information flowing between other nodes, while harmonic centrality indicates the shortest "distance" between a node and all other nodes in the graph. Since the control logic in a design is in charge of sending signals to control the data-path blocks, the betweenness centrality and harmonic centrality is expected to be higher for the state registers than the data registers.

Once the training set is created, the next step is to train the graph learning pipeline. The ReIGNN framework processes the graph data and generates node embeddings. To facilitate generalization across unseen graphs, we use GraphSAGE [20] as the GNN layer, which follows an inductive approach to generate node embeddings. Instead of training a distinct embedding vector for each node, GraphSAGE trains an update function that learns to aggregate feature information from a node's local neighborhood to generate node embeddings. Therefore, GraphSAGE simultaneously learns the topological structure of each node's neighborhood and the distribution of node features in the neighborhood [20].

We leverage structural information associated with the gate-level netlist graphs to further improve the classification accuracy of ReIGNN. The state registers in the control logic of a design must have a feedback path and they should all be part of a strongly connected component (SCC). After the GNN prediction, we use this information by checking if a register identified by the GNN as a state register is part of an SCC and has a feedback loop. If any of these criteria is not satisfied, we safely reclassify the register as a data register.

IV. Evaluation Results

A. Evaluation of the ML-ILP Attack

Our attack was evaluated on 19 ISCAS'89 and ITC'99 benchmarks locked by LBLL. To attack each of the benchmark circuits, we trained a model using the 18 other circuits and their variants, splitting samples between training and validation. The number of latch samples used for training the models ranged from 75k to 77k. We configured the ILP to search for the top 10k potential keys with the assumption that the attacker can individually test each of these keys. Even for the largest circuit tested, the ILP completes its search in less than one hour, finding the secret keys for 8 of the 19 locked circuits with 100% accuracy. Moreover, for all tested circuits, the average difference between the best key found and the correct key is 2.0%. To the best of our knowledge, this is the first successful attack on LBLL circuits, and is the first successful ML-based attack on any sequential logic locking technique.

B. Evaluation of ReIGNN

We assess the performance of ReIGNN on a set of standard benchmarks from OpenCores, the `secworks` GitHub repository, and blocks from a 32-bit RISC-V processor and a 32-bit microcontroller. While creating the dataset, we considered two versions of each design, following a one-hot encoding and a binary encoding for the FSM states. To evaluate the performance of ReIGNN, we compare its predictions with the true labels of the nodes and use the standard statistical measures of correctness of a binary classification test. Since the dataset is imbalanced, we use the *sensitivity*, or true positive rate (TPR), *specificity*, or true negative rate (TNR), and *balanced accuracy* metrics, defined below:

$$\text{Sensitivity} = \frac{\text{No. true positives}}{\text{No. true positives + No. false negatives}}, \quad (1)$$

$$\text{Specificity} = \frac{\text{No. true negatives}}{\text{No. true negatives + No. false positives}}, \quad (2)$$

$$\text{Balanced Accuracy} = \frac{\text{Sensitivity + Specificity}}{2}. \quad (3)$$

High sensitivity and high balanced accuracy are both desirable, since they impact any FSM extraction methodology for IC reverse-engineering. If we report low sensitivity, then there exist unidentified state registers, which will impact the correctness of the extracted FSM. On the other hand, low specificity, hence low balanced accuracy, makes the FSM extraction time consuming since many data registers are also identified as state registers and will be included in the state transition graph of the FSM.

We compared the performance of ReIGNN with the RELIC algorithm [17]. Across the one-hot encoded benchmarks, ReIGNN reports an average balanced classification accuracy of 94.7% and a sensitivity of 96.3%, to be contrasted with the average balanced classification accuracy of 86.6% and sensitivity of 88% reported by RELIC. Across the binary encoded benchmarks, ReIGNN reports an average balanced accuracy of 97.9% and a sensitivity of 99.4%, while RELIC achieves an average balanced accuracy of 86% and a sensitivity of 85.7%. Moreover, both balanced accuracy and sensitivity in RELIC show a significantly larger variability across different benchmarks based on the threshold values used to determine the similarity between registers. By combining graph neural networks with structural analysis, ReIGNN can classify the registers in a circuit with high accuracy and generalize well across different designs.

V. Conclusions

The effectiveness of the ML-ILP attack and ReIGNN approaches exemplifies that combining data (via trained ML classifiers) and structure (constraining the solution space to

feasible solutions) is a powerful approach for security attacks that must be considered when building any defense. Interesting future work includes more actively using the structural constraints during the training of the deep learning models, as in [21].

ACKNOWLEDGMENT

This material is based on research sponsored by the Air Force Research Labs (AFRL) and the Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-18-1-7817. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the Air Force Research Labs (AFRL), the Defense Advanced Research Projects Agency (DARPA), or the U.S. Government.

REFERENCES

[1] K. Shamsi, M. Li, K. Plaks, S. Fazzari, D. Z. Pan, and Y. Jin, "IP Protection and Supply Chain Security through Logic Obfuscation: A Systematic Overview," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 24, no. 6, pp. 1–36, 2019.

[2] D. Sisejkovic, L. M. Reimann, E. Moussavi, F. Merchant, and R. Leupers, "Logic Locking at the Frontiers of Machine Learning: A Survey on Developments and Opportunities," *arXiv preprint arXiv:2107.01915*, 2021.

[3] P. Chakraborty, J. Cruz, and S. Bhunia, "SAIL: Machine Learning Guided Structural Analysis Attack on Hardware Obfuscation," in *2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, 2018, pp. 56–61.

[4] D. Sisejkovic, F. Merchant, L. M. Reimann, H. Srivastava, A. Hallawa, and R. Leupers, "Challenging the Security of Logic Locking Schemes in the Era of Deep Learning: A Neuroevolutionary Approach," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 17, no. 3, pp. 1–26, 2021.

[5] J. Sweeney, V. Mohammed Zackriya, S. Pagliarini, and L. Pileggi, "Latch-Based Logic Locking," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2020, pp. 132–141.

[6] S. Dutta Chowdhury, K. Yang, and P. Nuzzo, "ReIGNN: State register identification using graph neural networks for circuit reverse engineering," in *Proc. Int. Conf. Computer-Aided Design*, 2021, pp. 1–9.

[7] K. Georgila, "Using Integer Linear Programming for Detecting Speech Disfluencies," in *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, 2009, pp. 109–112.

[8] T. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[9] R. Torrance and D. James, "The state-of-the-art in IC reverse engineering," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2009, pp. 363–381.

[10] S. E. Quadir, J. Chen, D. Forte, N. Asadizanjani, S. Shahbazmohamadi, L. Wang, J. Chandy, and M. Tehranipoor, "A survey on chip to system reverse engineering," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 1, pp. 1–34, 2016.

[11] J. Baehr, A. Bernardini, G. Sigl, and U. Schlichtmann, "Machine learning and structural characteristics for reverse engineering," *Integration*, vol. 72, pp. 1–12, 2020.

[12] A. Fayyazi, S. Shababi, P. Nuzzo, S. Nazarian, and M. Pedram, "Deep learning-based circuit recognition using sparse mapping and level-dependent decaying sum circuit representations," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 638–641.

[13] W. Li, Z. Wasson, and S. A. Seshia, "Reverse engineering circuits using behavioral pattern mining," in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2012, pp. 83–88.

[14] A. Gascón, P. Subramanyan, B. Dutertre, A. Tiwari, D. Jovanović, and S. Malik, "Template-based circuit understanding," in *Formal Methods in Computer-Aided Design (FMCAD)*. IEEE, 2014, pp. 83–90.

[15] T. Meade, S. Zhang, and Y. Jin, "Netlist reverse engineering for high-level functionality reconstruction," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2016, pp. 655–660.

[16] M. Fyrbiak, S. Wallat, J. Déchelotte, N. Albartus, S. Böcker, R. Tessier, and C. Paar, "On the difficulty of FSM-based hardware obfuscation," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 293–330, 2018.

[17] T. Meade, Y. Jin, M. Tehranipoor, and S. Zhang, "Gate-level netlist reverse engineering for hardware security: Control logic register identification," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2016, pp. 1334–1337.

[18] M. Brunner, J. Baehr, and G. Sigl, "Improving on state register identification in sequential hardware reverse engineering," in *IEEE Int. Symp. Hardware Oriented Security and Trust (HOST)*, 2019, pp. 151–160.

[19] J. Geist, T. Meade, S. Zhang, and Y. Jin, "RELIC-FUN: logic identification through functional signal comparisons," in *ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[20] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 1025–1035.

[21] A. Ferber, B. Wilder, B. Dilkina, and M. Tambe, "MIPaaL: Mixed integer program as a layer," in *AAAI Conference on Artificial Intelligence*, 2020.