Flip-Lock: A Flip-Flop-Based Logic Locking Technique for Thwarting ML-based and Algorithmic Structural Attacks

Armin Darjani
armin.darjani@tu-dresden.de
TU Dresden
Germany

Nima Kavand
nima.kavand@tu-dresden.de
TU Dresden
Germany

Akash Kumar
akash.kumar@ruhr-uni-bochum.de
Ruhr University Bochum
Germany

ABSTRACT

Machine learning (ML) and algorithmic structural attacks have highlighted the possibility of utilizing structural leakages of an obfuscated circuit to reverse engineer the locking mechanism. These structural leakages are rooted in security-agnostic synthesis tools that lead to discernible patterns within the vicinity of the locking substructures. This paper has two contributions. Firstly, we present the innovative Flip-lock, a novel approach that utilizes flip-flops along logic gates to prevent synthesis tools' structural leakages. As this is the first work that incorporates flip-flops in locking structures, our second contribution is the development of a comprehensive analysis tool designed to identify and assess potential structural leakages in the vicinity of flip-flops within circuits. We name our tool Flip-attack. We employ the Flip-attack analysis to strengthen Flip-lock's resilience against potential future attacks. Our findings demonstrate that Flip-lock possesses the capability to neutralize all existing state-of-the-art structural attacks effectively. Furthermore, by enhancing Flip-lock through the incorporation of our analysis tool, we establish a robust defense mechanism that can safeguard the circuit's security against potential future threats.

CCS CONCEPTS

• **Security and privacy → Security in hardware**.

KEYWORDS

Logic Locking, Structural attacks, ML-based attacks, Design-for-trust, Reverse engineering

ACM Reference Format:
Armin Darjani, Nima Kavand, and Akash Kumar. 2024. Flip-Lock: A Flip-Flop-Based Logic Locking Technique for Thwarting ML-based and Algorithmic Structural Attacks. In *Great Lakes Symposium on VLSI 2024 (GLSVLSI '24), June 12–14, 2024, Clearwater, FL, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3649476.3658753

1 INTRODUCTION

Logic locking is the most prominent and holistic DFT approach that offers protection through multiple stages of the IC supply chain [17]. By locking the design and adding new logic elements, this technique hides the true functionality of the circuit from an adversary. The design can only function correctly by feeding a valid key set to the circuit stored in a tamper-proof memory. Logic locking techniques can be categorized into gate and interconnect/routing obfuscation. Interconnect obfuscation techniques lock the netlist by intertwining multiple paths coming from various logic cones in a locking structure [19] using multiplexers. In gate obfuscation techniques, the designer perturbs the netlist by injecting locking gates (like XOR/XNOR) into the netlist [17].

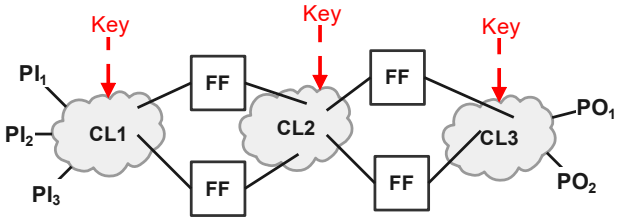


Figure 1: Conventional Logic locking in a sequential circuit

Although deemed as a safe technique, proposing seminal Boolean Satisfiability (SAT)-based attack [20] challenged the security of logic locking. This powerful attack utilizes a working duplicate of the target circuit (an Oracle) to infer the valid key. Despite the SAT-attack's initial success, works like [8, 18] showed that routing obfuscation techniques [18] or AND-tree structures [8] can thwart the SAT attack by increasing the run-time of the attack exponentially. Moreover, the threat model and efficiency of the SAT attack have been questioned. The main assumption in the threat model of many SAT attacks is access to an Oracle with an open scan chain in test mode, especially against large circuits with thousands of keys [13], which is not possible as all vendors close the access to the scan chain. Besides, accessing the oracle is almost impossible for very sophisticated specific circuits designed for the military, nuclear power plants, and other critical infrastructures. Regarding efficiency, attacking circuits with complex arithmetic units (cryptographic modules, 32-bit multipliers, etc.) is infeasible for the SAT attack [10, 12]. In addition to SAT attacks, research such as [15] highlights the vulnerability of logic locking to probing attacks when access to an Oracle is available. However, employing tamper-resistant packaging and designs, as suggested by [14] along with leveraging emerging technologies, as discussed in [16], can effectively thwart probing attacks.

Knowing the shortcomings of the SAT attacks, researchers have started to propose structural attacks [1, 3–5]. These attacks omit the need for an Oracle from the threat model; hence, they are called Oracle-less attacks. They utilize the structural features of the design to trace back the changes introduced into the circuit by logic

GLSVLSI '24, June 12–14, 2024, Clearwater, FL, USA

ACM ISBN 979-8-4007-0605-9/24/06
https://doi.org/10.1145/3649476.3658753

locking. There are four main advantages of structural attacks over SAT attacks. Firstly, they omit the need for an Oracle, leading to a more powerful threat model [1]. Secondly, they can break logic locking in circuits with SAT-hard functions [2]. thirdly, they are a better choice for attacking circuits with complex arithmetic units where SAT fails to infer the valid key set [10]. Finally, attackers can use structural attacks, employing a divide-and-conquer strategy to target the locked combinational parts of very large sequential circuits separately. This means each locked structure can be analyzed based on the characteristics of its surrounding structure. **Therefore, structural attacks pose potential threats in logic-locking scenarios where SAT and probing attacks are not feasible.**

The structural attacks revealed that although adding locking structures and re-synthesizing the design can lead to changes in a netlist, these modifications are readily detectable and reversible using sophisticated algorithms or ML models [1, 3, 5]. Furthermore, the scope of these changes is small and confined to the adjacency of the locked structures [4] that decreases the complexity of key classification to only small subgraphs of the circuit.

The origin of these structural vulnerabilities is rooted in the security-agnostic synthesis tools. In logic locking, the security designers start with a synthesized circuit. They add a locking structure to the circuit and re-synthesize the design using synthesis tools to intertwine the locking structure with the original design. These tools only consider power, area, and performance constraints in synthesizing a circuit. To this aim, Synthesis tools use a finite number of rules during circuit synthesis [5]. This results in observable behavior and structural traces surrounding newly introduced locking circuitry, such as multiplexers and XOR/XNOR gates. Attackers can exploit these traces to effectively reverse engineer the locked circuit back to its original design. Moreover, as the attacker has access to the circuit's netlist, she can re-synthesize the design multiple times using various tools and technology libraries that can highlight more structural traces around locking structures.

In this paper, we propose Flip-lock, which for the first time, incorporates flip-flops into the locking structures as a locking element to neutralize the structural leakages of the security-agnostic tools. Until this point, all structural attacks have focused on treating the targeted circuit as entirely combinational. The rationale behind this consideration is that any circuit can be divided into sequential and combinational parts, and key inputs are fed directly to the combinational portions, as illustrated in Figure 1. So, each combinational portion can be separately analyzed and reversed to the original structure. Presenting Flip-lock, we challenge traditional assumptions and demonstrate that integrating flip-flops into the locking mechanism enhances structural security within the circuit. The goal of the Flip-lock is to hide the locking keys inside the design by adding locking flip-flops to the circuit that are not discernible from the original flip-flops of the circuit. Flip-lock adds new locking flip-flops and locks a portion of the original flip-flops of the design. By doing this, firstly, Flip-lock stops the synthesis tools from propagating structural changes from the locking structure to the locked signal. Secondly, it deprives the attacker of any useful information even after multiple attempts at re-synthesizing using various tools, as flip-flops cannot be translated into another form of logic or structure. We show that Flip-lock completely omits the structural leakages caused by security-agnostic synthesis tools, thereby completely thwarting all state-of-the-art structural attacks. As this is the first work that considers flip-flops as locking elements, we provide a comprehensive security assessment tool for analyzing potential structural leakages surrounding locking structures containing flip-flops. We name our assessment tool Flip-attack, and we show that naive implementation of the Flip-lock leads to a few structural leakages that are not related to security-agnostic synthesis tools. We utilized this information to provide an enhanced straightforward implementation of the Flip-lock that safeguards the design from future potential attacks. The contributions of this paper are as follows:

- We propose Flip-lock, a novel logic locking technique that utilizes flip-flops as part of the locking structure, and we analyze the security of Flip-lock against various classes of state-of-the-art structural attacks.
- We propose a comprehensive analysis tool designed to identify and assess potential structural leakages in the vicinity of flip-flops within circuits. We call this tool Flip-attack, which utilizes the power of ML (Machine learning) to distinguish locking flip-flops from the normal flip-flops in a circuit. We use this tool to strengthen the Flip-lock to address potential future structural attacks.
- We provide a comprehensive security and overhead analysis of Flip-lock using ISCAS-89 and ITC-99 benchmarks, and we show that Flip-lock scheme decreases the accuracy and precision of state-of-the-art structural attacks to less than 50%, rendering them no more effective than random guessing while it avoids new potential leakages.

2 BACKGROUND AND RELATED WORKS

2.1 Structural attacks against logic locking

Algorithmic attacks: These attacks are only available for interconnect obfuscation. In a naive interconnect obfuscation, incorrectly setting a key value results in circuit reduction [19]. Constant propagation attacks like SCOPE [1] exploit this weakness and attack the circuit by hard-coding the value of one key bit at a time and performing re-synthesis. These attacks gather design features, including power, area, number of AND gates in AIG representation, and more, during the re-synthesizing phase by initializing each key bit separately to constant values one and zero. Finally, the attack correlates extracted features to correct key values.

ML-based attacks: These attacks harness the power of ML approaches like Graph Neural Networks (GNNs) to learn the impact of the security add-ons on their nearby subgraphs. These attacks are available for both gate and interconnect obfuscation. In SAIL [5], the authors showed that re-synthesizing a gate obfuscated circuit leads to local changes that can be learned and used to trace back the changes and finally infer the correct key values. Although SAIL is a powerful attack, it suffers from drawbacks like the need for pre-synthesis localities and complex learning models. In OMLA [3], authors showed that utilizing GNNs compensates for these drawbacks. OMLA maps the problem of resolving the key-bit value to subgraph classification. OMLA follows a self-referencing model for learning biases and behavior of the target netlist after adding

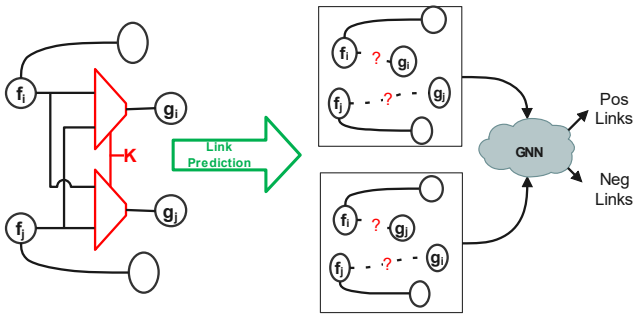


Figure 2: The flow of the MUXLink attack [4]

locking gates (XOR/XNORs) and re-synthesizing, resulting in significantly more accurate results.

[4] proposed the MUXLink attack for breaking interconnect obfuscation. MUXLink works based on the assumption that because of using security-agnostic synthesis tools, modern IC designs largely consist of repetitive structures. Harnessing the GNN, MUXLink learns the structures of unobfuscated parts of the locked design. The learned model will be utilized to deobfuscate the locked structures of the design using link prediction methods. Figure 2 shows the flow of the MUXLink attack. This attack first converts the netlist into a graph. Then, it omits the connections between target nodes for each D-MUX structure from the graph representation. Then, MUXLink extracts the h-hop enclosing subgraphs around the target nodes and assigns an 8-bit one-hot encoded vector and a tag to each node in the subgraphs. Here, the one-hot encoder contains information about the type of gate, and the tag contains relationship information between the node in the subgraph and the target nodes. This tag is calculated by the double radius node labeling (DRNL) equation. The target links are $(f_i, g_i), (f_i, g_j), (f_j, g_i), (f_j, g_j)$. The MuxLink trains a GNN on the unobfuscated interconnects, and by learning the link connections in circuit subgraphs, it makes predictions on the target links. The absolute difference δ between the likelihood scores for all its possible links is computed for each locked structure. As long as δ exceeds the classification threshold, the link with the highest likelihood score is predicted to exist.

2.2 Structural defenses

Previously, researchers have proposed some solutions to address the structural vulnerabilities of logic looking. [11] proposes eFPGA Fabrics for Logic Locking. This is basically a LUT-based logic locking technique that transfers a part of interconnects and logic to an on-chip embedded FPGA and programs this on-chip FPGA for the circuit activation. Although it can thwart structural attacks, it incurs a very high overhead to the circuit. Besides, incorporating an embedded FPGA is not possible for many circuits. Moreover, this defense mechanism is susceptible to predictive attacks [6].

[7, 9] propose new gate selection approaches for interconnect obfuscation that can thwart structural attacks against interconnect obfuscation. However, these techniques limit the security designers in the locking process regarding both the number of keys and signal selection. Moreover, these works have not evaluated the protection levels after multiple circuit re-synthesizing. Furthermore, these works do not discuss thwarting ML-based attacks against gate obfuscation, limiting the designer only to interconnect obfuscation.

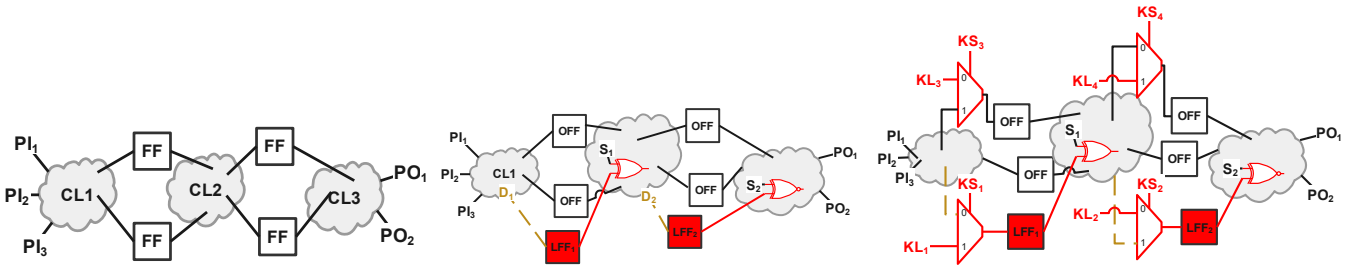
The shortcomings of the available structural defenses point out the need for a novel locking mechanism. This mechanism should provide fine-grain access to the circuit's logic gates; it should not need any add-on embedded on-chip modules like eFPGAs and should not limit the designer regarding the locking mechanism (gate or interconnect obfuscation), number of key bits, and signal selection. In the following section, we propose the Flip-lock, a novel logic locking technique with none of the aforementioned shortcomings.

3 FLIP-LOCK

Flip-lock adds new locking flip-flops to the circuit that store and feed locking keys to the combinational parts of the circuit. We call these new locking flip-flops LFFs. To hide the LFFs, Flip-lock also selects a portion of the original flip-flops of the circuit (OFFs) and locks them using a multiplexer.

Figure 3 shows the steps Flip-lock takes for securing a circuit. Flip-lock starts with a sequential circuit containing combinational and sequential elements. Then, it selects pairs of (D_i, S_i) signals. Here, D_i is a single signal, whereas S_i can be a set of signals containing more than one signal from the combinational parts of the circuit. In a naive implementation, D_i and S_i can be selected arbitrarily. However, in Section 4, we will show that a naive pair selection might lead to a sub-optimal design regarding security, and we will propose a pair selection approach. After selecting (D_i, S_i), Flip-lock adds an LFF to the circuit. It connects D_i to the input of the LFF and treats the output of this LFF as a key bit connected to S_i. This key bit can lock one or multiple signals using available gate or interconnect obfuscation techniques. The key value for each LFF should be selected by the designer. For example, in Figure 3b, the designer decides that LFF_1 stores and outputs a key value of $Zero$ in each cycle where LFF_2 stores and outputs a key value of One. Flip-lock uses the outputs of the LFFs to lock S_i signals.

After adding LFFs, Flip-lock re-synthesizes the design. This step optimizes the design for newly added LFFs. A naming convention separates LFFs from OFFs in the design so that the designer can still identify the LFFs inside the netlist after re-synthesizing. At the final phase, Flip-lock selects a portion of OFFs and starts adding multiplexers to selected OFFs and all the LFFs like Figure 3c. This way, the LFFs are not distinguishable from the locked OFFs. In the design illustrated in Figure 3c, The multiplexers connected to the selected OFFs always select the original signals coming from the combinational part using selection keys we name as KS_i whereas the multiplexers connected to the LFFs select locking key inputs KL_i. Note that although, after re-synthesizing, the structure around the LFFs changes, the logic function still remains the same. So, feeding correct values to the LFFs still leads to correct functionality. The value of a KL_i connected to a multiplexer of an OFF can be set to both 0 or 1 as it will never be used. The output of LFFs can be used as a key for obfuscating the combinational part of the circuit using a gate or interconnect obfuscation. For example, in Figure 3, we only use the output of LFFs for gate obfuscation using additional XOR/XNOR logic cells to lock S_is. The output of LFFs can also be used as a selector of a locking multiplexer that obfuscates the paths inside the combinational parts of the circuit like [19]. In design shown in Figure 3c, for selectors we have



(a) Original circuit (b) LFF addition (c) Final locked circuit

Figure 3: Flip-lock flow. (a) the original circuit, (b) the circuit after adding LFFs and locking structures (XOR/XNOR gates), (c) the Final Flip-locked circuit

$\{KS_1, KS_2, KS_3, KS_4\} = \{1, 0, 1, 0\}$ and for locking keys we have $\{KL_1, KL_2, KL_3, KL_4\} = \{0, 1, X, X\}$ where X can be 1 or 0.

Note that care should be taken to preserve the functionality of the circuit. Since the correct outputs of the LFFs are available after the first clock cycle, all the S_is should be situated within level 2 of the circuit and beyond (CL2, CL3, and others). Consequently, at least one of the inputs of the locked gates must originate from a flip-flop. Subsequently, after the first cycle, the output of the *LFFs* remains correct for the remainder of the circuit's operation. Note that the outputs of the locked OFFs are always correct, even for the first cycle, as the locked OFFs are fed with the original signal from the circuit using KS_is.

Breaking a circuit secured by the Flip-lock involves a two-stage process. Initially, the task is to separate the LFFs and locked OFFs. During this phase, the attacker's objective is to retrieve the values of KS_is. Subsequently, leveraging structural attack methods becomes viable for extracting the values of KL_is associated with the LFFs. Regarding state-of-the-art structural attacks, retrieving the value of KS_is can be mapped to the problem of breaking interconnect obfuscation. Each multiplexer is fed by two different paths. One is a key input (KL_i) coming from the tamper-proof memory, and the other comes from the combinational logic. Regarding distinguishing LFFs from locked OFFs, we analyze the constant propagation and ML-based link prediction attacks. For the constant propagation attacks, Flip-lock completely omits the problem of circuit reduction [1]. The D_i connected to the LFFs are considered a part of the original design by constant propagation attacks, so they wrongly consider all the LFFs as OFFs of the design. For the ML-based link prediction attacks against interconnect obfuscation, Flip-lock breaks the structural relation between the input signals of a locking multiplexer with the locked signals at the output of the multiplexer using flip-flops. The synthesis tools will not propagate the optimization from the input of the flip-flop to the output of the flip-flop. As a result, MUXLink cannot find any relation between D_i and S_i. Regarding ML-based gate obfuscation attacks, the attacker needs first to distinguish the LFFs from locked OFFs. These attacks cannot distinguish LFFs and OFFs because they are not designed for such locking schemes, and secondly, re-synthesizing the circuit will lead to the same structure. So, these attacks cannot pinpoint and break the locking gates (the XOR and XNOR connected to LFF_1 and LFF_2 in Figure 3). In Section 5, we analyze the security of the Flip-lock against state-of-the-art structural attacks.

4 SECURITY ANALYSIS TOOL FOR FLIP-FLOP LOCKING

Flip-lock thwarts structural attacks against logic locking by eliminating leakages related to synthesis tools. However, utilizing flip-flops can lead to new structural traces. This section analyzes potential structural leakages in a circuit locked with Flip-lock. We propose the Flip-attack that shows that naive implementation of Flip-lock might lead to very few structural traces. We propose a straightforward algorithm to avert these structural leakages.

4.1 Threat model

The attacker has access to the full netlist of the locked circuit, but she has no access to the functional chip (Oracle). The attacker knows the circuit is locked using the naive Flip-lock technique, where (D_i, S_i) pairs are selected randomly from the combinational parts of the circuit. Moreover, the locked OFFs are randomly selected. The attacker has no prior information regarding the number of LFFs and OFFs as it is part of the information embedded in *KS*s.

4.2 Flip-attack

Flip-attack aims to classify locked flip-flops of the circuit as LFFs and OFFs. This attack is a self-reference ML-based attack. We use a self-referencing model as we observed that using a generic model leads to low accuracy as flip-flops of each circuit can have different surrounding substructures. This aligns with observations in [3, 4]. This attack first converts the circuit to a directed graph and an undirected graph with the logic gates and flip-flops as nodes and gate connections as edges. We call the directed graph *DiGraph* and the undirected graph *UGraph*. Utilizing the *DiGraph*, we extract key structural features for each flip-flop node, specifically the outdegree (IN_{od}) of the node connected to the flip-flop and the flip-flop's own outdegree (FF_{od}). From the *UGraph*, we extract the shortest path from the flip-flop input node to the flip-flop output nodes, excluding the direct link between the flip-flop and its input. We call this feature *SP*. Additionally, we capture the *Graphlet_count* feature, representing the count of graphlets of size 3 in a subgraph within a 3-hop radius around each flip-flop node. A graphlet is a small, connected subgraph within a larger graph. It's essentially a small pattern or motif that appears in the overall structure of the graph. IN_{od} and FF_{od} capture the immediate connections of the flip-flop's input and output, illustrating its direct relationships. Meanwhile, *Graphlet_count* explores the complexities of a 3-hop subgraph wrapping the flip-flop, offering insights into the broader

Algorithm 1 Flip-attack's training algorithm

1: **INPUT:** $Locked\ UGraph,\ Locked\ DGraph$
2: **OUTPUT:** $Trained\ Classifier$
3: $Features = \{IN_{od}, FF_{od}, graphlet\_count, SP\}$
4: $Pos\_vector = extract\_features(OFFs, UGraph, DGraph)$
5: $Ne\_vector = extract\_features(LFFs, UGraph, DGraph)$
6: $Features = RFECV(classifier, Features, Pos\_vector, Neg\_vector)$
7: $Prune(Pos\_vector)$
8: $Prune(Neg\_vector)$
9: $Classifier.train(Pos\_vector, Neg\_vector)$

structure of the subgraph. Lastly, SP encapsulates the relationship between the flip-flop's input and output(s).

To analyze the potential leakages of the naive Flip-lock, first, we train a classifier by randomly adding LFFs using the naive Flip-lock technique and comparing the features of LFFs with available OFFs in the circuit. We extract $\{IN_{od}, FF_{od}, Graphlet\_count, SP\}$ for each new LFF we have added to the circuit. We also extract the mentioned features for all available OFFs. Here, LFFs are negative samples, whereas OFFs are positive samples. We use Recursive Feature Elimination with Cross-Validation (RFECV) to find each circuit's best set of features. RFECV recursively removes features, fitting a model and evaluating its performance through cross-validation. This iterative process helps identify the optimal subset of features that maximizes predictive performance. Finally, we train a classifier using a pruned feature set. Algorithm 1 depicts the training process of Flip-attack.

We carry out the Flip-attack for multiple circuits. Our first observation was the importance of the IN_{od}. The RFECV returns only the IN_{od} for all the benchmarks. This indicates that, in the majority of cases, the input of each flip-flop is not connected to any other sequential or logic elements as an input. So, selecting D_i from available combinational logic almost always leads to the Flip-attack's success. To omit such an obvious leakage, we change the naive implementation of the Flip-lock. We first make an arbitrary dummy signal by adding a new logic gate to the circuit and connecting its input(s) to arbitrary signals from the combinational part of the circuit. We connect the output of the dummy gate to the LFF. We call this gate a dummy gate, as its output will never be used in the circuit. After incorporating this modification, we initiated the training phase for the circuit. RFECV consistently avoided selecting IN_{od} once more during our observations, as we anticipated. Moreover, the $Graphlet\_count$ feature was never selected by the RFECV. This observation underscores the absence of discriminative structures around OFFs and LFFs. To reinforce this claim, we show the ineffectiveness of the MUXLink attack against Flip-lock in Section 5. Since MUXLink employs a robust GNN approach to identify structural patterns, its failure indicates the lack of discernible patterns around the OFFs that could distinguish them from LFFs. By omitting the IN_{od} and $Graphlet\_count$, the two remaining features, FF_{od} and SP, were always selected as discriminative features by RFECV. Given these insights, we present an enhanced Flip-lock in Algorithm 2 aiming to mitigate potential structural leakages of Flip-lock.

Algorithm 2 Enhanced Flip-lock

1: **INPUT:** $Netlist,\ key_pairs$
2: **OUTPUT:** $Locked\ circuit$
3: $dic\_FF_{od} = Extract\_FF_{od}(Netlist)$
4: $SP_{min}, SP_{med} = Extract\_min_med(Netlist)$
5: **while** $ACC > 60$ **do**
6: $key_{count} = 0$
7: **while** $key_{count} < |key\_pairs|/2$ **do**
8: $sig_{pair} = randomSel(Netlist)$
9: $D_i = newGate(sig_{pair})$
10: $S_i = Select(Netlist, D_i, dic\_FF_{od}, SP_{min}, SP_{med})$
11: $LFFs = AddLFFs(Netlist, D_i, S_i)$
12: $Key_{count} = Key_{count} + 1$
13: **end while**
14: $LockedOFFs = LockOFFs(Netlist, |key\_pairs|/2)$
15: $ACC = Accuracy(LockedOFFs, LFFs, Classifier)$
16: **end while**

The input of the enhanced Flip-lock algorithm is the number of key pairs (KS_i, KL_i) and the netlist of the circuit. The output of this algorithm is a secured netlist. This algorithm starts with determining the optimal number of LFFs' outputs (FF_{od}). For that, we first extract FF_{od} for all the OFFs in the circuit and the number of repetitions for each value. We store this information in a dictionary called dic_FF_{od}. To determine optimal values for LFFs' SP feature, we extract the SP for all OFFs and calculate the minimum and median of the OFFs' SP. We store this information in SP_{min} and SP_{med}. Now, the algorithm can start the selection phase. In our algorithm, 50% of the key pairs (KS_i, KL_i) are used for locking OFFs, and 50% of them are used for adding LFFs. However, a security designer can change this ratio based on overhead and security needs. The algorithm starts with the netlist of the circuit. It selects two arbitrary signals that are close to each other in the $UGraph$ of the netlist. The algorithm adds a new logic gate to the circuit using arbitrary signals as its inputs. This signal is the new D_i (line 9). When selecting the S_i the algorithm utilizes the information available in SP_{med}, SP_{min}, and dic_FF_{od} (line 10). This process will be repeated until the intended number of LFFs are added to the circuit. After adding LFFs, we randomly select a portion of OFFs to lock. After the selection phase, we evaluate the security of the locked circuit using a Flip-attack. For this, we use our new approach for training the Flip-attack using multiple LFFs and OFFs based on the size of the circuit. The goal is an accuracy of less than 60% for the Flip-attack. If the accuracy is higher than 60% the selection algorithm is repeated.

5 EVALUATION

5.1 Experimental setup

For the evaluation of the security and overhead of Flip-lock, we use 6 circuits from ISCAS-89 and ITC-99 benchmark sets. In our Flip-lock implementation, we use half of the keys for locking OFFs and the other half for adding LFFs. Table 1 contains the information about these circuits. As we mentioned, breaking Flip-lock can be mapped to breaking an interconnect obfuscation scheme. So, We use the open-source available MUXLink [4] and SCOPE [1] attacks

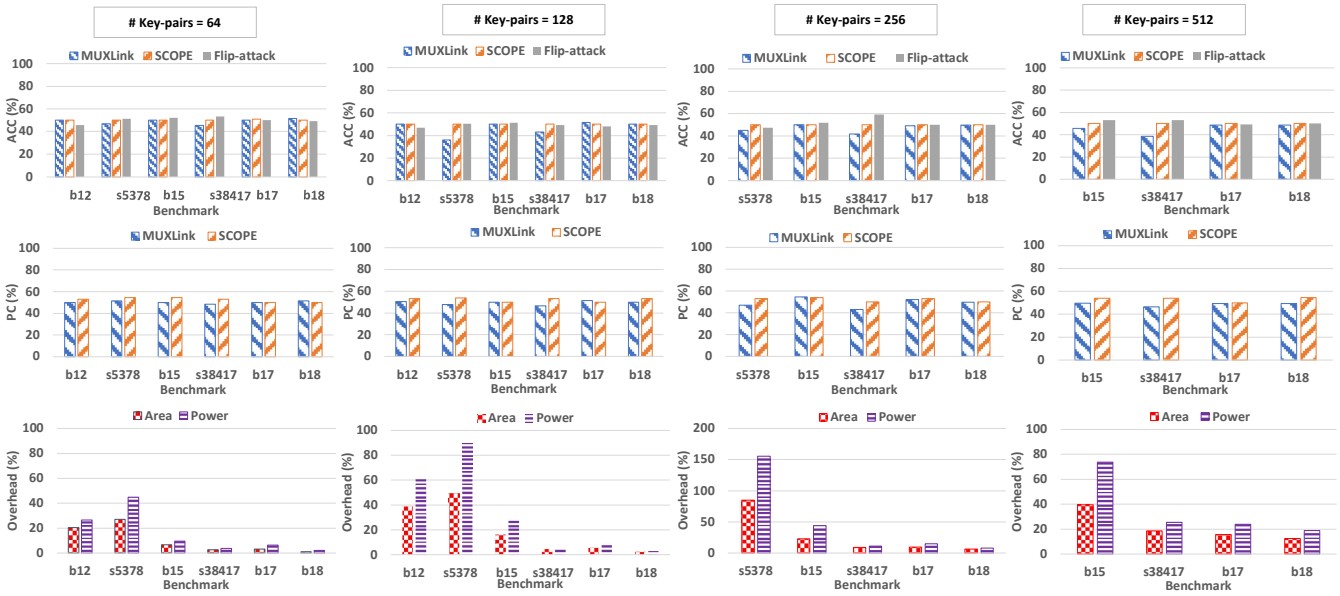


Figure 4: Security and overhead analysis of the Flip-lock for various key-pair sizes

with their default options. We employed Cadence GENUS for synthesizing the circuits, utilizing the 45nm open cell Nangate library to assess the area and power overhead of the Flip-lock. The synthesis and subsequent attacks were conducted on a high-performance server with 64 cores (AMD EPYC) and 128GB of RAM.

5.2 Flip-lock analysis

Security analysis: For assessing the security of the Flip-lock against various attacks, we use the same accuracy (AC) and precision metrics (PC) that are employed to show the MUXLink and SCOPE attacks' strength [1, 4]. These metrics have been shown in Eq. 1 and 2 respectively. The accuracy calculates the ratio of corrected guessed keys. In both MUXLink and SCOPE attacks, if a key value cannot be predicted, the attacks return X, which means the value is unknown. A higher PC indicates an increased likelihood of accurately guessing a key value through the attack. Conversely, a PC of 50% or lower translates to a complete attack failure, as it imparts inaccurate information (wrongly guessed key values) for more than half of the keys.

$$AC = \frac{KS_{(Correct)}}{KS_{(Total)}} * 100\% \quad (1)$$

$$PC = \frac{KS_{(Correct)} + KS(X)}{KS_{(Total)}} * 100\% \quad (2)$$

Breaking the Flip-lock involves a two-stage process. In the first stage, the attacker must classify locked flip-flops as OFFs and LFFs by guessing the KS_i values. The second stage requires guessing the KL_i values for the identified LFFs. If the attacker cannot complete the first stage, the second stage becomes unfeasible. For this, we report the AC and PC for only selecting keys (KSs) in Eq. 1 and 2.

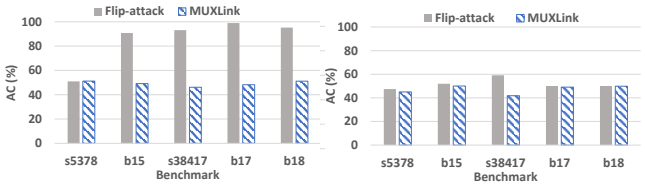
Figure 4 shows the AC and PC values for MUXLink, SCOPE, and Flip-attack against the circuits locked with the enhanced Flip-lock shown in Algorithm 2. For the MUXLink attack, we include the flip-flops as a new gate type. Now, the problem of breaking the Flip-lock can be mapped to a link prediction problem where MUXLink should select the correct value for each KS_i. The SCOPE attack was designed to extract useful structural information by re-synthesizing the circuit under attack for each key bit. For SCOPE, we cannot consider flip-flops as logic gates because, for most of the circuits, it leads to a combinational loop that leads to synthesis tools errors. For that, we investigate the SCOPE attack by unrolling the flip-flops of the design for one clock cycle. We create an unrolled version of each circuit and run the SCOPE attack on the fully combinational unrolled circuit.

Table 1: Benchmark Information

Benchmark	# Logic Gates	# Inverters	# Flip-flops
b12	963	113	121
s5378	1004	1775	179
b15	7922	1000	449
s38417	8709	13470	1636
b17	27852	4474	1415
b18	94249	20372	3320

As mentioned, we use half of the key pairs (KS_i, KL_i) for locking OFFs and the other half for adding LFFs. If a circuit lacks sufficient OFFs for a specific key size, we do not consider that circuit. As illustrated in Figure 4, Flip-lock completely thwarts all the attacks, including the novel Flip-attack with the AC and PC of less than 60%. *The inability of the MUXLink and Flip-attack underscores the absence of distinctive structural traces around LFFs and OFFs that can be leveraged to categorize these locked flip-flops. Meanwhile, the SCOPE attack's failure emphasizes that re-synthesizing a Flip-lock-secured circuit provides no useful information for the attacker.*

Robustness of our assessment tool: As we mentioned for the results we provided in Figure 4, we used the enhanced Flip-lock technique proposed in Algorithm 2 that was designed to omit structural leakages around the flip-flops.



(a) Naive Flip-lock (b) Enhanced Flip-lock

Figure 5: The comparison between security of the naive and enhanced Flip-lock implementation

To provide insights regarding the robustness of our security assessment tool, Flip-attack, and also show the ineffectiveness of state-of-the-art structural attacks even against the naive Flip-lock implementation, we compare the AC of these attacks for a naive and enhanced implementation of the Flip-lock in Figure 5 for various circuits locked with 256 key-pairs. In the naive implementation, we select 128 OFFs randomly and lock them. Then, we add 128 LFFs, adding a new gate as D_i connected to two arbitrary inputs. The output of the LFFs (S_i) is also arbitrary. For the enhanced Flip-lock, we use the results we provided in Figure 4.

To provide a comprehensive analysis, we provide an explanation for each individual circuit. For $S5378$, all or most of the original flip-flops will be locked, So there are no or very few OFFs left in the circuit for the Flip-attack's training phase. This limitation of the self-referencing model is also mentioned in [9]. However, as we mentioned, a generic training model leads to very low accuracy as various circuits have various structures [3, 4]; So we do not use a generic model. For other circuits, we can see that Flip-attack can yield high accuracy in breaking naive Flip-lock (Figure 5a); however, our enhanced Flip-lock can completely thwart this attack as shown in Figure 5b. Furthermore, our analysis highlights the MUXLink attack's incapacity to break even a naive Flip-lock implementation. This underscores the effectiveness of even a naive Flip-lock in nullifying the capabilities of state-of-the-art structural attacks to extract the key from surrounding substructures.

Overhead analysis: Figure 4 illustrates our Flip-lock implementation's area and power overheads. The results indicate that Flip-lock introduces a manageable overhead [19] relative to the circuit sizes. Notably, even with 512 key pairs, the overhead for medium and large circuits remains acceptable. Notably, Flip-lock incurs overheads of less than 10% for medium and large circuits, even in implementations with a substantial key-pair size of 256. This shows that Flip-lock can protect complex commercial circuits with acceptable overheads.

5.3 A discussion about flip-flop selection strategy for Flip-lock

For clarity, in all our implementations of the Flip-lock, we consider a 50 − 50 ratio for OFFs and LFFs. However, as outlined in our threat model, this ratio remains unknown to potential attackers since it is embedded in the KS_i. This presents a potent trade-off for the security designer concerning the overhead and security of the locked circuit. For instance, the Flip-lock can be configured to exclusively lock the OFFs of the circuit, omitting any addition of LFFs. Since the attacker lacks prior knowledge of the KS_is and does not have access to the Oracle, breaking the circuit remains impossible. This represents the Flip-lock implementation with the least possible overhead, as locking an OFF only requires a multiplexer. On the other end of the spectrum, all the locked flip-flops can be LFFs. This grants the designer the option of strategic placement of LFFs for locking intended gates. A designer can improve our Flip-lock algorithm to encompass various strategies regarding the attacking environments.

6 SUMMARY

In this paper, we have presented the Flip-lock, a novel logic-locking technique that utilizes flip-flops along with logic gates to lock the circuit. The goal of the Flip-lock is to hide the locking keys inside the design using locking flip-flops that are not discernible from the original flip-flops of the circuit. This way, Flip-lock completely omits the structural leakages regarding synthesis tools. As Flip-lock is the first locking scheme entangling flip-flops in the locking structure, we developed Flip-attack, a comprehensive analysis tool designed to identify and assess potential structural leakages in the vicinity of flip-flops within circuits. Our findings demonstrate that enhanced Flip-lock not only neutralizes existing structural attacks but also fortifies the circuit against potential future threats.

REFERENCES

[1] Abdulrahman Alaql et al. 2021. SCOPE: Synthesis-based constant propagation attack on logic locking. *TVLSI* (2021).
[2] Lilas Alrahis et al. 2021. GNNUnlock: Graph neural networks-based oracle-less unlocking scheme for provably secure logic locking. In *DATE*.
[3] Lilas Alrahis et al. 2021. OMLA: An oracle-less machine learning-based attack on logic locking. *TCAS-II* (2021).
[4] Lilas Alrahis et al. 2022. MuxLink: Circumventing learning-resilient MUX-locking using graph neural network-based link prediction. In *DATE*.
[5] Prabuddha Chakraborty et al. 2018. SAIL: Machine learning guided structural analysis attack on hardware obfuscation. In *AsianHOST*.
[6] Prattay Chowdhury et al. 2022. Predictive model attack for embedded fpga logic locking. In *Symposium on Low Power Electronics and Design*.
[7] Subhajit Dutta Chowdhury et al. 2023. Similarity-Based Logic Locking Against Machine Learning Attacks. In *DAC*.
[8] Armin Darjani et al. 2022. ENTANGLE: an enhanced logic-locking technique for thwarting SAT and structural attacks. In *GVLSI*.
[9] Armin Darjani et al. 2023. Discerning the Limitations of GNN-Based Attacks on Logic Locking. In *DAC*.
[10] Hadi Mardani Kamali et al. 2022. Advances in logic locking: Past, present, and prospects. *Cryptology ePrint Archive* (2022).
[11] Hadi M Kamali et al. 2023. SheLL: Shrinking eFPGA Fabrics for Logic Locking. In *DATE*. IEEE.
[12] Likhitha Mankali et al. 2022. Titan: Security Analysis of Large-Scale Hardware Obfuscation Using Graph Neural Networks. *TIFS* (2022).
[13] Abdul Moosa et al. 2023. Scaling Attacks on Large Logic-Locked Designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2023).
[14] Mosavirik et al. 2022. Impedanceverif: On-chip impedance sensing for system-level tampering detection. *IACR* (2022).
[15] M Rahman et al. 2020. The key is left under the mat: On the inappropriate security assumption of logic locking schemes. In *HOST*. IEEE.
[16] Nikhil Rangarajan et al. 2021. Tamper-proof hardware from emerging technologies. *The Next Era in Hardware Security: A Perspective on Emerging Technologies for Secure Electronics* (2021).
[17] Jarrod A Roy et al. 2008. EPIC: Ending piracy of integrated circuits. In *Proc. of the conference on Design, automation and test in Europe*.
[18] Akashdeep Saha et al. 2020. Lopher: Sat-hardened logic embedding on block ciphers. In *DAC*.
[19] Dominik Sisejkovic et al. 2021. Deceptive logic locking for hardware integrity protection against machine learning attacks. *TCAD* (2021).
[20] Pramod Subramanyan et al. 2015. Evaluating the security of logic encryption algorithms. In *HOST*.