IsoLock: Thwarting Link-Prediction Attacks on Routing Obfuscation by Graph Isomorphism

Shaza Elsharief, Lilas Alrahis, Johann Knechtel and Ozgur Sinanoglu

New York University Abu Dhabi, Abu Dhabi, UAE, {se1525,lma387,jk176,os22}@nyu.edu

Abstract.
Logic locking/obfuscation secures hardware designs from untrusted entities throughout the globalized semiconductor supply chain. Machine learning (ML) recently challenged the security of locking: such attacks successfully captured the locking-induced, structural design modifications to decipher obfuscated gates. Although routing obfuscation eliminates this threat, more recent attacks exposed new vulnerabilities, like link formation, breaking such schemes. Thus, there is still a need for advanced, truly learning-resilient locking solutions.
Here we propose *IsoLock*, a provably-secure locking scheme that utilizes isomorphic structures which ML models and other structural methods cannot discriminate. Unlike prior work, IsoLock's security promise neither relies on re-synthesis nor on dedicated sub-circuits. Instead, IsoLock introduces isomorphic key-gate structures within the design via systematic routing obfuscation. We theoretically prove the security of IsoLock against modeling attacks. Further, we lock ISCAS-85 and ITC-99 benchmarks and launch state-of-the-art ML attacks, SCOPE and MuxLink, as well as the Redundancy and SAAM attacks, which only decipher an average of 0–6% of the key, well confirming the resilience of IsoLock. All in all, IsoLock is proposed to break the cycle of "cat and mouse" in locking and attack studies, through a provably-secure locking approach against structural ML attacks.

Keywords: Hardware security · IP piracy · Graph neural networks · Logic locking · Machine learning

1 Introduction

The continued rise of cost and complexities for integrated circuits (ICs) manufacturing has shifted the semiconductor industry toward a horizontal model,[1] where design houses like Apple® outsource fabrication to *possibly untrusted* chip manufacturers [AM13]. Such globalized IC supply chain has given rise to a plethora of security threats, like hardware Trojans, counterfeiting, and intellectual property (IP) piracy. Several *design-for-trust* methods seek to regain trust in the supply chain and thwart IP piracy: layout camouflaging, split manufacturing, and logic locking/obfuscation [HCS+21].

Logic locking is a prominent solution as it aims to protect the design intellectual property throughout the semiconductor supply chain. It obfuscates both the structure and functionality of a design by integrating key-controlled logic elements, referred to as *key-gate structures* (or key-gates for short, which is apt especially for simple locking schemes using only simple gates). These key-gate structures bind the correct functionality of the design to a secret key that is only known to the legitimate IP owner. Thus, during outsourced design and fabrication stages, the design is not revealed in full. The IP owner, in coordination

[1]For example, commissioning a new 3nm facility costs TSMC® at least \$23 billion [Zaf]. Further, the slowdown in scaling, as seen for 5nm downto 3nm, demonstrates ever-increasing manufacturing complexities [Zaf22].

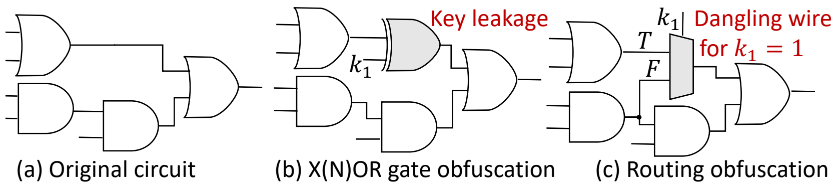

Figure 1: Different logic-obfuscation key-gate structures. The secret key is $k_1 = 0$ for all instances. T and F denote true and false/dummy interconnects.

with other trusted parties, loads the secret key into an on-chip, tamper-proof memory after fabrication and testing. Figure 1 (a) demonstrates an example circuit, which is locked using an XOR key-gate in Figure 1 (b) versus using a multiplexer (MUX) in Figure 1(c).

Researchers have evaluated logic locking mainly under two threat models, i.e., *oracle-guided attacks* [SRM15] and *oracle-less attacks* [LO19, APSS21, ARB21]. In the oracle-guided model, adversaries have access to a functional chip holding the key, the *oracle*, which serves to provide correct input-output patterns that are used for iterative procedures inferring the key by, e.g., principle of exclusion for mismatching observations. In contrast, the oracle-less model relies only on the netlist of the locked design to decipher the secret key [LO19] or to remove the protection logic altogether [APH+21a].

In this work, we focus on the oracle-less model, which is more powerful and also more realistic; the latter because it can be launched easily throughout any stage in the supply chain. In the oracle-less setting, machine learning (ML) has proven effective in learning the structure of locked circuits and breaking logic locking [CCAB21, APSS21, SMR+21, APSS22]. As a result, researchers have been motivated to devise learning-resilient locking schemes. However, the existing obfuscation schemes suffer from at least one of the limitations illustrated in Table 1 and discussed next.

1.1 Motivation and Research Challenges

Key Leakage: In traditional X(N)OR locking [RKM10], there is a direct mapping between the key-gate structures and the corresponding key-bits: XOR key-gates require key-bit '0' whereas XNOR key-gates require key-bit '1'. Re-synthesis performs logical and structural transformations, with the aim of obfuscating this mapping. Nevertheless, ML attacks such as SAIL [CCAB21], SnapShot [SMR+21], and OMLA [APSS21] succeeded to learn such synthesis-induced modifications around these key-gates and to still decipher the key, demonstrating that logic synthesis is deterministic and can be largely undone using ML.

Circuit Reduction: In MUX-based routing obfuscation, a MUX key-gate takes in a true (T) wire and a false/dummy (F) wire; only the correct key passes the T wire to the MUX output. Note that the T wire can be easily connected/routed to either the first or second input of the MUX and, thus, the correct key can be tailored to be '0' or '1'. Though free of key leakage, a naive implementation of such MUX obfuscation can still result in other structural hints, exploited by attacks based on constant propagation like SCOPE [ARB21] and SWEEP [AFB19] and by attacks based on structural analysis like SAAM [SMRL21]. Figure 1 (c) shows an example where the wrong key ($k_1 = 1$) results in a dangling T wire, which would be superfluous from a design perspective, indicating that this wire is indeed the T wire.

Design, Synthesis Dependency: Truly-random logic locking (TRLL) is proposed as learning-resilient, X(N)OR-based locking scheme [LKK+20]. However, the work in [SMRL21] shows that TRLL fails an "AND-netlist test",[2] indicating that TRLL de-

[2]This test explores the learning resilience of any locking scheme, by applying the scheme to an example circuit consisting only of AND gates, e.g., an AND-tree netlist. After locking, the test studies any structural changes, whether they can be predicted and correlated to the underlying key-bits [SMRL21]. TRLL fails

Table 1: Limitations of Logic Obfuscation Exposed by ML Attacks

Limitation / **Defense**	**Key Leakage**	**Circuit Reduction**	**Synthesis Depend.**	**Link-Form. Leakage**	**Fails AND-Netlist Test**
Tradit. X(N)OR [RKM10]	Yes	No	Yes	No	Yes
Tradit. MUX [RZZ+15]	No	Yes	No	Yes	No
UNSAIL [APK+21]	No	No	Yes	No	Yes
TRLL [LKK+20]	No	No	No	No	Yes
D-MUX [SMRL21]	No	No	No	Yes	No
Symm. MUX [ARB21]	No	No	No	Yes	No
Proposed IsoLock	**No**	**No**	**No**	**No**	**No**

pends on design properties, i.e., it is not a generally resilient solution. UNSAIL [APK+21] relies on targeted re-synthesis to generate obfuscation structures that confuse ML attacks. Ideally, however, security guarantees of locking should be independent of synthesis, which could introduce new vulnerabilities on its own, as its correct-by-construction, but not secure-by-construction.

Link-Formation Leakage: Deceptive MUX (D-MUX) [SMRL21] and symmetric MUX obfuscation [ARB21] circumvented all existing ML attacks prior to their development. Both schemes employ MUX key-gates to eradicate key leakage and carefully select the T/F wires to eliminate structural hints. However, MuxLink [APSS22] exposed a new vulnerability, i.e., link formation, breaking both schemes. MuxLink learns the composition of gates in a given design and deciphers the inputs of the MUXes, by solving a link-prediction problem using a graph neural network (GNN).

Research Challenge: Our analysis shows that there is still a shortfall in learning-resilient obfuscation solutions.[3] For example, while using MUX key-gates eliminates the issue of key leakage, there is no provably secure technique yet for selecting T/F wires without exhibiting link-formation leakage.

1.2 Our Novel Concept and Contributions

To address the above challenge, we propose *IsoLock*, a provably-secure routing obfuscation scheme that is resilient to link-prediction attacks. IsoLock introduces isomorphic routing solutions as outlined in Figure 2. Note that IsoLock is a generic obfuscation methodology. Here, we demonstrate the effectiveness of IsoLock for MUX-based locking, but IsoLock can be easily applied to any other routing obfuscation such as intercorrelated logic and routing locking (InterLock) [KAHS20] and interconnect camouflaging [PASK20, CCW18]. For IsoLock, the wires introduced by routing obfuscation are all truly equally likely, without inducing any structural hints or information leakage, thwarting all existing oracle-less ML attacks on obfuscation. Our contributions in this work are as follows:

1. **Isomorphic Obfuscation (Section 3).** After converting a gate-level netlist to a graph, IsoLock systematically searches the graph for isomorphic structures suitable for locking. IsoLock then integrates, without loss of generality (w/o.l.o.g), MUX key-gates on those isomorphic structures without structural information leakage on the secret key.

the test as there are no inverters to begin with, which are used by TRLL for key-gate structures.

[3]Other works may become relevant toward that end as well. For example, LoPher is a locking scheme that inherits the security properties of block ciphers, by employing cipher-like substitution boxes for obfuscation. While LoPher was specifically proposed to hinder oracle-guided attacks [SSC+20], one can argue that LoPher might be secure against structural attacks as well, given the regular structures of substitution boxes. However, this has not been demonstrated yet in [SSC+20]. In any case, LoPher has been considered (so far) only for protecting small and selected sub-circuits, not a whole design, due to its prohibitive overheads [SSC+20].

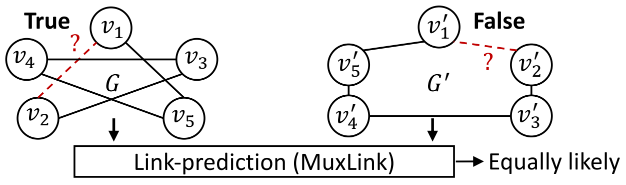

Figure 2: High-level concept of IsoLock. The subgraphs G and $G^{'}$ extracted around links obfuscated by IsoLock (red dashed links) are isomorphic, yet have distinct labels (true/false) considering the target links. However, link-prediction attacks (and other structural attacks) cannot distinguish between the two subgraphs—all target links are equally likely, providing secure obfuscation.

Table 2: Commonly Used Notations

Notation	Definition	Notation	Definition
$\mathcal{G}$	Graph	n	Number of vertices in $\mathcal{G}$
V	Set of vertices in $\mathcal{G}$	E	Set of edges in $\mathcal{G}$
$\mathbf{A}$	Adjacency matrix of $\mathcal{G}$	T	Set of target links
$S = \{u, v\}$	Denotes the link between u, v	$\mathcal{G}_{(S,h)}$	h-hop enclosing subgraph
$\mathcal{N}(v)$	Direct neighbors of v	v	A vertex in V
S_{sel}	Selected MUX locking strategy	$\mathbf{z}_v^{(l)}$	Embedding of v at l-th GNN layer
$d(u, v)$	Shortest-path distance b/w vertices u, v	π	A permutation
$\mathbf{a}_v^{(l)}$	Output of GNN aggregation	L	Total number of GNN layers
th	MuxLink classification threshold	K	Key-size
h'	Attack hop-size	h	Locking hop-size
f_1, f_2	Input gates to lock	g_1, g_2	Output gates to lock
F_{multi}	Set of multi-output vertices in $\mathcal{G}$	F_{single}	Set of single-output vertices in $\mathcal{G}$

2. **Mathematical Proof (Section 3.1).** We present a formalism to prove, and thus, guarantee link-prediction resilience.

3. **Implementation (Section 3.2, 3.3).** As the subgraph isomorphism problem is NP-complete, we identify some relaxed, yet appropriate, assumptions that allow for an efficient implementation. We will release IsoLock and its artifacts post peer-review.

2 Background and Related works

Next, we provide the background related to D-MUX [SMRL21], link-prediction, GNNs, and MuxLink [APSS22], which all are relevant for understanding our proposed IsoLock scheme. For ease of reference, we summarize the commonly used notations in Table 2.

2.1 Deceptive MUX Logic Locking (D-MUX)

For IsoLock, we build on the D-MUX scheme [SMRL21], enhancing it to thwart the MuxLink attack. W/o.l.o.g., we focus on D-MUX as it offers inherent resilience against other structural attacks, mainly SAAM [SMRL21], with reasonable design overhead.

D-MUX works as follows. First, all gates in the netlist are split into two sets, namely *multi-output gates* versus *single-output gates*. Second, a locking strategy is chosen to lock two gates $\{f_1, f_2\}$; this step is repeated until the desired key-size is achieved. The process for gate selection is summarized in IsoLock's Algorithm 2, lines 1–9. D-MUX [SMRL21]

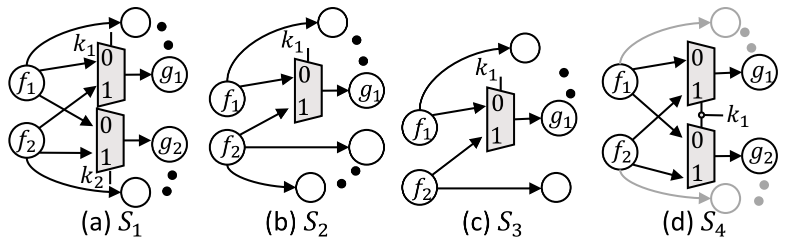

Figure 3: The D-MUX locking strategies [SMRL21]. Vertices/edges represent gates/wires.

selects the T/F connections based on multiple locking strategies S_1–S_4 (Figure 3), which are explained next.

- The S_1 strategy is to randomly select two multi-output gates as inputs for two locking MUXes, controlled by two (interdependent) key-inputs $\{k_1, k_2\}$. That is, each MUX locks one randomly selected output gate $g \in \{g_1, g_2\}$ using both input gates $\{f_1, f_2\}$ for routing obfuscation.
- The S_2 strategy is to randomly select two multi-output gates but integrate them using a single MUX, locking one output gate for a randomly selected input gate.
- The S_3 strategy is to randomly select a multi-output gate f_1 and a single-output gate f_2, then integrate them using a single MUX, locking one of the outputs of f_1.
- Finally, the S_4 strategy sets no restrictions on $\{f_1, f_2\}$. Thus, two MUXes, controlled by the same key-input, are used to lock one output gate for each input gate.

Locking strategies are picked based on the availability of single-output and multi-output gates in the design. All strategies avoid creating combinational loops in the locked design. Since the S_4 strategy is always applicable, any desired (yet practical, e.g., without inducing loops) key-size can be achieved. For more details on the strategies, please also refer to [SMRL21].

2.2 Link-Prediction Problem

Link prediction has diverse applications, such as friend suggestion in social networks [AA03] and protein-interaction prediction [QBJKS06], to name a few. Link-prediction algorithms estimate the likelihood of a link to form between two target vertices in a given network based on the structure of the network and the properties of its vertices [ZC18].

Let $\mathcal{G} = (V, E, \mathbf{A})$ denote an n-vertex graph, where V is the set of vertices, $E \subseteq V \times V$ is the set of observed edges, and $\mathbf{A} \in \{0,1\}^{n \times n}$ is the adjacency matrix, where $\mathbf{A}_{i,j} = 1 \iff (i,j) \in E$. Link-prediction algorithms assign *likelihood scores* $\in [0,1]$ to links of interest, i.e., *target links* T, where $T \notin E$. A target link is denoted by the vertex set $S \in V$, which includes the two end vertices of the link.

In GNN-based link-prediction, an *h-hop enclosing subgraph* is extracted around each target link. These subgraphs contain information about the vertices surrounding the links. Thus, by virtue of performing graph classification, the labels of subgraphs also become the labels for the enclosed target links [ZC18].

Definition 1. h-Hop Enclosing Subgraph: Given $(S, \mathcal{G})$, the $\mathcal{G}_{(S,h)}$ subgraph is induced from $\mathcal{G}$ by $\cup_{v \in S}\{u \mid d(u,v) \leq h\}$, where $d(u,v)$ is the shortest-path distance between vertices u and v. $\mathbf{A}_{(S,h)}$ denotes the adjacency matrix of $\mathcal{G}_{(S,h)}$.

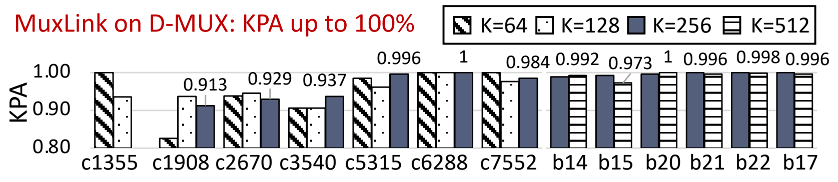

Figure 4: D-MUX [SMRL21] locking is vulnerable to MuxLink [APSS22]. Results are based on own experiments using the framework provided in [APSS22]. MuxLink achieves a key-prediction accuracy (KPA) of 100%, deciphering the whole key.

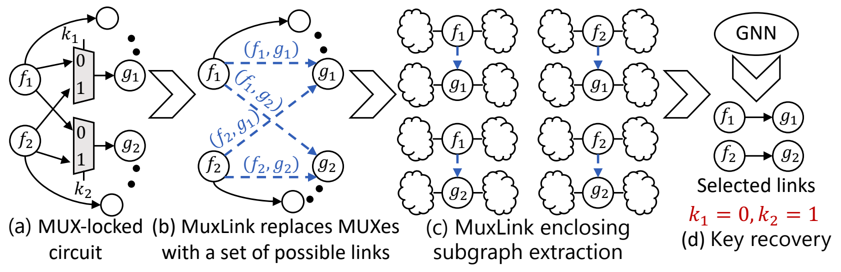

Figure 5: The high-level concept of MuxLink [APSS22].

In general, GNNs operate on graph structured data and generate a vector representation, i.e., *embedding*, for each vertex in the graph. Embeddings are to be used for a desired task, such as vertex, graph, or link classification. Embedding generation includes L rounds of *neighborhood aggregation*, also known as *message passing*. For each round, each vertex $v \in V$ updates its embedding based on the information received from its neighboring vertices. A single aggregation round l is abstracted as follows, where $\mathbf{z}_v^{(l)}$ indicates the embedding of vertex v at the l-th round and $\mathcal{N}(v)$ indicates the direct neighbors of v.

$$\mathbf{a}_v^{(l)} = AGG^{(l)}\left(\left\{\mathbf{z}_u^{(l-1)} : u \in \mathcal{N}(v)\right\}\right) \quad (1)$$

$$\mathbf{z}_v^{(l)} = UPDATE^{(l)}\left(\mathbf{z}_v^{(l-1)}, \mathbf{a}_v^{(l)}\right) \quad (2)$$

More specifically, for each round, the collected information is first aggregated using a AGG function (obtaining $\mathbf{a}_v^{(l)}$) and then combined with the features of the target vertex using an $UPDATE$ function [KW17]. Hence, the embedding of a vertex captures its properties, the properties of its neighborhood, and its integration within the graph. Different GNN implementations employ different AGG and $UPDATE$ functions.

For link-prediction in particular, a readout is applied over the vertex embeddings in the subgraph, to obtain a link-level embedding which is used for classification, i.e., predicting the existence versus non-existence of a link.

2.3 MuxLink

The MuxLink attack [APSS22] targets on the D-MUX locking scheme [SMRL21]. Although D-MUX showed good resilience against all existing oracle-less ML attacks at the time, MuxLink successfully circumvented it, as also demonstrated in Figure 4.

The intuition behind MuxLink is that modern IC designs are largely repetitive, based on reuse of circuit modules and on regular structures for functionalities like adders, etc. Thus, MuxLink employs a GNN to first learn the structures of unobfuscated parts of any locked design and then, using this domain knowledge, makes predictions on the obfuscated interconnects. Considering the example in Figure 5, MuxLink converts the locality in

Figure 5(a) into a graph with missing connections shown in Figure 5(b). Here, four different connections are possible depending on the key-bits; $(f_1, g_1), (f_1, g_2), (f_2, g_1), (f_2, g_2)$ are the target links. MuxLink then extracts h-hop enclosing subgraphs around the target links. Next, MuxLink trains a GNN on the un-obfuscated interconnects remaining in the design, learns the composition of gates, and makes predictions on the target subgraphs. For each MUX, the absolute difference δ between the likelihood scores for all its possible links is computed. If δ is greater than the *classification threshold th*, the link with the highest likelihood score is predicted to exist. Otherwise, MuxLink does not make a prediction for that specific locality. Finally, the predictions are evaluated for extraction of the key-bits, as shown in Figure 5(c). In short, for D-MUX (and other schemes), MuxLink can succeed whenever each of the extracted subgraphs in Figure 5(b) is different and, thus, will exhibit different likelihood values.

3 Proposed IsoLock Scheme

IsoLock utilizes the same locking strategies as D-MUX (Figure 3). Importantly, however, IsoLock selects the $\{f_1, f_2, g_1, g_2\}$ gates for locking in a more careful way, namely to generate four identical structures around the four possible connections for these gates. To obtain such isomorphic structures, we search for and lock *isomorphic target links.*

3.1 Proof of Resilience Against Learning

We define permutation and graph isomorphism next, required as first step to prove the resilience of IsoLock.

Definition 2. Permutation: A permutation π is a bijective function from set $\{1, 2, \ldots, n\}$ to itself. A permutation group is a group Π_n whose elements are all $n!$ possible π. For link prediction, $S = \{u, v\}$ denotes the link between u, v. We define $\pi(S) = \{\pi(u)|u \in S\}$. We further define the permutation of $\mathbf{A}$ as $\pi(\mathbf{A})$, where $\pi(\mathbf{A})_{\pi(u),\pi(v),:} = \mathbf{A}_{u,v,:}$.

Definition 3. Graph Isomorphism: Given two n-vertex graphs $\mathcal{G} = (V, E, \mathbf{A})$, $\mathcal{G}' = (V', E', \mathbf{A}')$ and two vertex sets $S \subseteq V$, $S' \subseteq V'$, we say $(S, \mathbf{A})$ and $(S', \mathbf{A}')$ are isomorphic – denoted by $(S, \mathbf{A}) \simeq (S', \mathbf{A}')$ – if $\exists \pi \in \Pi_n$ such that $S = \pi(S')$ and $\mathbf{A} = \pi(\mathbf{A}')$. When $(V, \mathbf{A}) \simeq (V', \mathbf{A}')$, we say two graphs $\mathcal{G}$ and $\mathcal{G}'$ are *isomorphic* – denoted as $\mathbf{A} \simeq \mathbf{A}'$, since $V = \pi(V')$.

The work in [ZC18] demonstrated that most successful link-prediction heuristics are approximated from local subgraphs. Therefore, in link-prediction attacks, an h-hop enclosing subgraph around each target link is extracted, restricting the GNN to perform neighborhood aggregation within the subgraph boundary [APSS22, APH+21b]. The larger the subgraph around the target link, the more information from the neighbourhood is collected; the latter does not correlate well with the localized link formation. That is, excess information expressed through larger subgraphs is not relevant and is typically even misleading. Accordingly, for IsoLock, we focus on *local h-isomorphism.*

Definition 4. Local h-Isomorphism: $\forall S, \mathbf{A}, S', \mathbf{A}'$, we say $(S, \mathbf{A})$ and $(S', \mathbf{A}')$ are locally h-isomorphic to each other if $(S, \mathbf{A}_{(S,h)}) \simeq (S', \mathbf{A}'_{(S',h)})$.

Lemma 1. *Given two h-isomorphic subgraphs around two target links, a GNN will predict the same probability for both subgraphs.*

Proof. Recall that, in the context of GNNs, the embedding of vertex v is generated as $\mathbf{z}_v^{(L)} = UPDATE^{(L)}\left(\mathbf{z}_v^{(L-1)}, \mathbf{a}_v^{(L)}\right)$. Therefore, when $\mathbf{z}_v^{(L-1)} = \mathbf{z}_u^{(L-1)}$ and $\mathbf{a}_v^{(L)} = \mathbf{a}_u^{(L)}$ for two vertices v and u, the embeddings of the two vertices are identical, i.e., $\mathbf{z}_v^{(L)} = \mathbf{z}_u^{(L)}$. $\square$

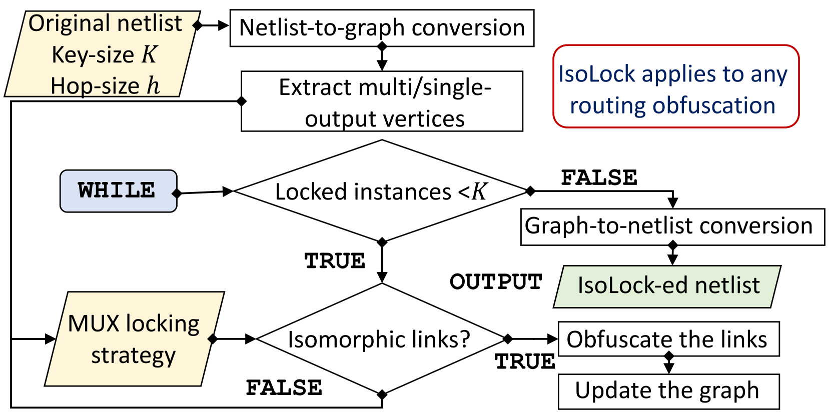

Figure 6: Integration of IsoLock with routing obfuscation.

In summary, **a GNN will learn the same vertex representation for all the vertices in two (or more) h-isomorphic graphs, given their identical neighborhoods.** For such subgraphs, all target links will be predicted with equal probabilities – if one link is predicted to exist, the other(s) should exist too, and vice versa. Thus, the proposed IsoLock scheme for routing obfuscation is proven to be learning-resilient.

3.2 Putting it All Together

The proposed IsoLock scheme is presented in Algorithm 1 and Figure 6. The input netlist is to be represented as a directed graph $\mathcal{G}$ with vertices and edges representing the gates and interconnects, respectively. The user also provides the desired key-size K, the locking strategies L_s, the locking hop-size h, etc. Recall that the strategies L_s are based on the D-MUX scheme [SMRL21]; also recall Figure 3.

In the main part of Algorithm 1, the locking strategies (with random selection among them) iteratively explore a set of possible links to lock, considering only links with isomorphic enclosing subgraphs. Note that the maximum-iteration variables related to input and output vertices, I_{max} and O_{max}, limit the number of re-selections explored for respective vertex pairs. Once the desired key-size is reached, the graph is updated based on all the locked localities and, finally, converted back to a gate-level netlist. We show an example of a simplified IsoLock-ed netlist in Figure 7.

Locking of isomorphic links (line 12 in Algorithm 1) is delegated to Algorithm 2. By default, the IsoCheck function in Algorithm 2 (line 19) implements the *VF2* graph isomorphism algorithm [CFSV01, FSV01]. VF2 builds up isomorphism one match (i.e., pair of vertices) at a time, using an educated backtracking search for matches. The VF2 algorithm uses effective data structures for storing information related to the state space search, significantly reducing the matching time and the memory requirements compared to other algorithms; please refer to [CFSV01, FSV01] for more details. Note that the IsoCheck function returns 'true' only if the corresponding subgraphs around the two target links are isomorphic. That is, IsoCheck($\mathcal{G}_{((f_1,g_1),h)}$,$\mathcal{G}_{((f_2,g_1),h)}$) returns 'true' only if $\mathcal{G}_{((f_1,g_1),h)} \simeq \mathcal{G}_{((f_2,g_1),h)}$. Also note that, for matters of practical implementation, relaxing the isomorphism check should be considered, as we discuss in Section 3.3.2.

Algorithm 1 IsoLock Locking Scheme

```
Input: Locking strategies $L_s$, key $K$, netlist graph $\mathcal{G}$, max input vertex iterations $I_{max}$, max output vertex
    iterations $O_{max}$, locking hop-size $h$
Output: Locked netlist
 1: $\{F_{single}, F_{multi}\} \leftarrow \{\emptyset\}$
 2: for $v$ in $\mathcal{G}$ do
 3:     if $|v.successors|>1$ then
 4:         $F_{multi} \leftarrow v$
 5:     else if $|v.successors| == 1$ then
 6:         $F_{single} \leftarrow v$
 7:     end if
 8: end for
 9: $Locked\_instances \leftarrow 0$
10: while $Locked\_instances < |K|$ do
11:     $S_{sel} \leftarrow$ RndPick($L_s$)                // IsoLock can take any MUX locking strategy
12:     $\{\{f_1, f_2\}, \{g_1, g_2\}, done\} \leftarrow$           // Search for Isomorphic links
                FindIsomorphism($S_{sel}, F_{single}, F_{multi}, I_{max}, O_{max}, \mathcal{G}, h$)
13:     if !$done$ then                                // Check if the search was successful
14:         continue
15:     end if
16:     $K_{1,2}, Locked\_instances \leftarrow$ GetFrom($K, S_{sel}$)     // Apply selected strategy
17:     $\{Lock_{1,2}\} \leftarrow$CoupleToMUXs($f_1, f_2, g_1, g_2, K_{1,2}$)
18:     UpdateNetlist($\mathcal{G}, \{Lock_{1,2}\}$)
19: end while
20: return $GraphToNetlist(\mathcal{G})$
```

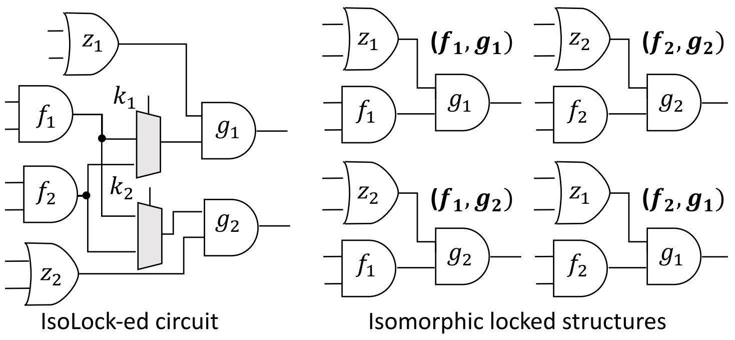

Figure 7: An IsoLock-ed gate-level netlist. The obfuscated links are isomorphic (i.e., equally likely). IsoLock does not add new subcircuits, rather it adds key-controlled MUXes to manipulate the original interconnects of the design.

3.3 Relaxed Assumptions for Practical IsoLock Implementations

3.3.1 Hop-Size

As we use local h-isomorphism in IsoLock, we must select a proper value h. One can argue that link-prediction attacks would want to increase the size of subgraphs extracted around target links as much as needed. For example, if two links are found to be isomorphic at $h = 3$, the attacker would want to check whether isomorphism still holds at $h = 4$. We set $h = 3$ by default for the reasons discussed next, and in our experiments (Section 4), we also demonstrate the diminishing effect of increasing h to 4.

Analyzing the number of GNN layers required to reach the theoretical limit of GNNs, it is recommended that any GNN-based prediction is restricted to a certain number of neighborhood aggregation layers. This problem is known as *over-smoothing* [LHW18]. In practice, target vertices will benefit the most from information gathered around their 3-hops neighborhood. As discussed in Section 3.1, the work in [ZC18] indeed demonstrated that misleading information may be included when considering subgraphs of size ≥ 3, thereby reducing the performance of their link-prediction attack.

Algorithm 2 IsoLock FindIsomorphism Function

```
Input: Strategy $S_i$, $F_{single}$, $F_{multi}$, $I_{max}$, $O_{max}$, $\mathcal{G}$, $h$
Output: Valid input vertices $\{f_2, f_1\}$, valid output vertices $\{g_2, g_1\}$, $done$
 1: $\{F_1, F_2\} \leftarrow \{\emptyset\}$
 2: if $S_i \in \{S_3\}$ then                              // Select vertices based on fan-out (D-MUX approach)
 3:     $\{F_1, F_2\} \leftarrow \{F_{multi}, F_{single}\}$
 4: else if $S_i \in \{S_1, S_2\}$ then
 5:     $\{F_1, F_2\} \leftarrow \{F_{multi}, F_{multi}\}$
 6: else
 7:     $\{F_1, F_2\} \leftarrow \{F_{single} \cup F_{multi}\}$
 8: end if
 9: $\{f_1, f_2, g_1, g_2\} \leftarrow \emptyset$; $done \leftarrow$ FALSE                  // Prepare link end vertices
10: for $iter_{in} = 0$ to $I_{max}$ do
11:     $\{f_1, f_2\} \leftarrow \{$RndPick($F_1$), RndPick($F_2$)$\}$)          // Start with a random selection
12:     while $f_1 == f_2$ do
13:         $\{f_1, f_2\} \leftarrow \{$RndPick($F_1$), RndPick($F_2$)$\}$)     // Make sure $f_1$ is different than $f_2$
14:     end while
15:     for $iter_{out} = 0$ to $O_{max}$ do
16:         $\{g_1, g_2\} \leftarrow \{$RndPick(successors($f_1$)), RndPick(successors($f_2$))$\}$      // Check for loops
17:         $\{Loop_1, Loop_2\} \leftarrow \{$IsInOutCone($f_2, g_1$), IsInOutCone($f_1, g_2$)$\}$
18:         if ($g_1$ != $g_2$) && !$Loop_1$ && !$Loop_2$ then              // Pass IsoLock requirements
19:             if IsoCheck($\mathcal{G}_{((f_1,g_1),h)}$,$\mathcal{G}_{((f_2,g_1),h)}$) && IsoCheck($\mathcal{G}_{((f_1,g_2),h)}$,$\mathcal{G}_{((f_2,g_2),h)}$)&&
            IsoCheck($\mathcal{G}_{((f_1,g_1),h)}$,$\mathcal{G}_{((f_1,g_2),h)}$)&& IsoCheck($\mathcal{G}_{((f_2,g_1),h)}$,$\mathcal{G}_{((f_2,g_2),h)}$) then
20:                 $done \leftarrow$ TRUE
21:                 break
22:             end if
23:         end if
24:     end for
25:     if $done$ then
26:         break
27:     end if
28: end for
29: return $\{\{f_1, f_2\}, \{g_1, g_2\}, done\}$
```

Algorithm 3 IsoLock Variant I - IsoCheck Function

```
Input: $\mathcal{G}$, $\mathcal{G}'$
Output: $IsoCheck$
 1: $IsoCheck \leftarrow$ FALSE
 2: if $\mathcal{G}.order() == \mathcal{G}'.order()$ then                     // Check local properties
 3:     $d_1 \leftarrow$ Sorted($d$ for $v$, $d \in \mathcal{G}.degree()$)       // An iterator for (vertex, degree) in $\mathcal{G}$
 4:     $d_2 \leftarrow$ Sorted($d$ for $v$, $d \in \mathcal{G}'.degree()$)
 5:     if $d_1 == d_2$ then
 6:         $IsoCheck \leftarrow$ TRUE
 7:     end if
 8: end if
 9: return $IsoCheck$
```

3.3.2 Isomorphism Check – IsoLock Variants

The graph isomorphism problem is known to be NP-complete. To speed up the locking procedure and achieve larger key-sizes, we explore relaxing the notion of isomorphism checks. We provide three variants of IsoLock, described next.

- **IsoLock Variant I (Algorithm 3):** This variant checks for matches in *degree sequences*, i.e., the list of degrees – number of incoming/outgoing edges – for all vertices in the graph, sorted in ascending or descending order. By construction, two isomorphic graphs will exhibit matching degree sequences.[4] Therefore, if two graphs have mismatching degree sequences, they cannot be isomorphic—this fact allows for a simple and fast exclusion check Note that, although two non-isomorphic graphs may exhibit the same degree sequence in some particular sorted order of vertices, such graphs tend to have a high structural similarity, which can be good enough to mislead the GNN, as we demonstrate through our experiments.

- **IsoLock Variant II (Algorithm 4):** This variant checks for matches in degree

[4]For example, the ascending-order sequence for the isomorphic graphs $\mathcal{G}$ and $\mathcal{G}'$ in Figure 2 is (1,1,2,2,2), assuming the target link in red does not exist.

Algorithm 4 IsoLock Variant II - IsoCheck Function

```
Input: $\mathcal{G}$, $\mathcal{G}'$
Output: $IsoCheck$
 1: $IsoCheck \leftarrow$ FALSE
 2: if $\mathcal{G}.order() == \mathcal{G}'.order()$ then                 // Check local properties
 3:     $d_1 \leftarrow \mathcal{G}.degree()$
 4:     $t_1 \leftarrow \mathcal{G}.triangles()$                     // Number of triangles keyed by vertex
 5:     $props_1 \leftarrow$ Sorted([$d$, $t_1[v]$] for $v$, $d$ in $d_1$)
 6:     $d_2 \leftarrow \mathcal{G}'.degree()$
 7:     $t_2 \leftarrow \mathcal{G}'.triangles()$                    // Number of triangles keyed by vertex
 8:     $props_2 \leftarrow$ Sorted([$d$, $t_2[v]$ for $v$, $d$ in $d_2$)
 9:     if $props_1 == props_2$ then
10:         $IsoCheck \leftarrow$ TRUE
11:     end if
12: end if
13: return $IsoCheck$
```

Algorithm 5 IsoLock Variant III - IsoCheck Function

```
Input: $\mathcal{G}$, $\mathcal{G}'$
Output: $IsoCheck$
 1: $IsoCheck \leftarrow$ FALSE
 2: if $\mathcal{G}.order() == \mathcal{G}'.order()$ then                 // Check local properties
 3:     $d_1 \leftarrow \mathcal{G}.degree()$
 4:     $t_1 \leftarrow \mathcal{G}.triangles()$                     // Number of triangles keyed by vertex
 5:     $c_1 \leftarrow \mathcal{G}.cliques()$               // Number of maximal cliques for each vertex
 6:     $props_1 \leftarrow$ Sorted([$d$, $t_1[v]$, $c_1[v]$] for $v$, $d$ in $d_1$)
 7:     $d_2 \leftarrow \mathcal{G}'.degree()$
 8:     $t_2 \leftarrow \mathcal{G}'.triangles()$                    // Number of triangles keyed by vertex
 9:     $c_2 \leftarrow \mathcal{G}'.cliques()$              // Number of maximal cliques for each vertex
10:     $props_2 \leftarrow$ Sorted([$d$, $t_2[v]$, $c_2[v]$] for $v$, $d$ in $d_2$)
11:     if $props_1 == props_2$ then
12:         $IsoCheck \leftarrow$ TRUE
13:     end if
14: end if
15: return $IsoCheck$
```

sequences and *triangle sequences*; this is more strict than Variant I. The triangle value of a vertex $v \in \mathcal{G}$ is the number of triangles in $\mathcal{G}$ that include v as one vertex. The triangle sequence is the list of triangles for all the vertices in the graph, sorted in ascending or descending order. By construction, two isomorphic graphs will exhibit matching triangle sequences. For example, the ascending-order triangle sequence for the isomorphic graphs $\mathcal{G}$ and $\mathcal{G}'$ in Figure 2 is (0,0,0,0,0).

- **IsoLock Variant III (Algorithm 5):** This variant checks for matches in degree and triangle sequences as well as *clique sequences*; this is more strict than both Variant I and Variant II. A clique is a subset of vertices $C \in V$ of $\mathcal{G}$ such that the subgraph build up by C is fully connected. That is, every distinct pair of vertices in C are connected by a distinct edge of $\mathcal{G}$. The clique number of a vertex $v \in \mathcal{G}$ is the number of vertices in the *maximal cliques* in $\mathcal{G}$ including v. A maximal clique is a clique in $\mathcal{G}$ that cannot be extended by including one more adjacent vertex, meaning it is not a subset of a larger clique. By construction, two isomorphic graphs will exhibit matching clique sequences. For example, the ascending-order clique sequence for the isomorphic graphs $\mathcal{G}$ and $\mathcal{G}'$ in Figure 2 is (1,1,2,2,2), assuming the target link in red does not exist.

Recall that the IsoLock scheme is proven to be learning resilient in general. We refer to the related, VF2-based implementation of IsoLock as *TrueIso*. Once the isomorphic check is relaxed, however, the proof does not hold anymore. Accordingly, the above IsoLock Variants I, II, and III are not proven to be 100% learning resilient.

Still, all variants do represent a complex problem for ML, effectively thwarting related structural attacks. We demonstrate this claim in detail in Section 4. With the practical implementation provided through these three different variants, we enable the user to select from a trade-off range for achievable key-sizes and runtimes required for locking, all while maintaining a demonstrably high learning resilience across the variants.

4 Evaluation

4.1 Setup

4.1.1 Experimental Setup

We implement IsoLock using *Python*, and we utilize the *NetworkX* library's isomorphic check functions. For benchmarks, we use the *ICAS-85* and *ITC-99* combinational suites. For logic synthesis, we use the *NanGate 45nm Open Cell Library*.

W/o.l.o.g., we consider key-sizes of 16, 32, 64, and 128 bits. IsoLock structures are iteratively embedded, following Section 3, until desired key-sizes or the maximal possible size is reached. For the latter, we report results for the next-lower key-size as appropriate; e.g., if only 120 bits can be reached, not 128, we report results for 64 bits then.

For security evaluation, we utilize the following attacks, obtained as-is through the respective publications and weblinks: MuxLink [APSS22], SCOPE [ARB21], SAAM [SMRL21], and Redundancy [LO19]. For MuxLink, we consider hop-sizes h' of 3 and 4, as elaborated in Section 3.3.1. We also vary the classification threshold th, to explore its role on the attack performance. We use the default settings for SCOPE, SAAM, and Redundancy.

For IsoLock, we consider hop-sizes h of 1–4 independently, to explore the role of the size of isomorphic key-gate structures. Recall that we consider three IsoLock variants (Section 3.3.2) that utilize different relaxed versions of isomorphism checks, and we also consider the proven, unrelaxed isomorphism check via IsoLock TrueIso; all to study the impact of the isomorphism check on runtime, achievable key-sizes, and resilience of the proposed locking scheme. Throughout the reported experiments, we consider IsoLock Variant I as default option, unless stated otherwise.

MuxLink and IsoLock are run on an HPC cluster, mainly for parallelized batch processing, with 10 cores (Intel(R) Xeon(R) CPU E5-2680 v4 @2.4GHz) and 3.75G of RAM. SAAM, SCOPE, and Redundancy are run on a regular workstation with an Intel(R) Xeon(R) CPU X5680 @3.33GHz and 95GB of RAM. We consider a time-out of 48h for all attack runs as well as all IsoLock runs.

For layout evaluation, we use *yosys-abc* [W$^+$20] as follows. We employ a simple recipe based on tutorials [W$^+$20] and run it twice: first without timing constraint, second with the constraint reported by the first run. This allows us to determine a practical timing constraint for each design independently and further optimize the design accordingly.[5]

4.1.2 Metrics

For security evaluation, we use three metrics: *accuracy* (AC), *precision* (PC), and *key-prediction accuracy* (KPA) AC is defined as the ratio of correctly deciphered key-bits out of the full key: $(K_{correct}/K_{total})\cdot 100\%$. PC is defined as the ratio of correctly deciphered keys, optimistically counting every undecided bit X as correct guess: $((K_{correct}+K_X)/K_{total})\cdot 100\%$. KPA is defined as the ratio of correctly deciphered key-bits out of all predictions: $(K_{correct}/(K_{total}-K_X))\cdot 100\%$. All metrics are reported in %. For AC, PC, and KPA, in general, the lower the values, the less successful is the attack and the more resilient is the locking scheme. However, it is important to note that, in cases where both KPA and AC are 0%, a PC of 100% means that best resilience is achieved. This is because, in such

[5]We have also explored running more than two rounds, but found that results had largely saturated after the second round. Further, considering the resilience of IsoLock, since the employed key-gate structures are simple MUXes with (carefully selected) interconnects, these structures are *not* transformed or undermined by re-synthesis runs. We also verify this using custom scripts.

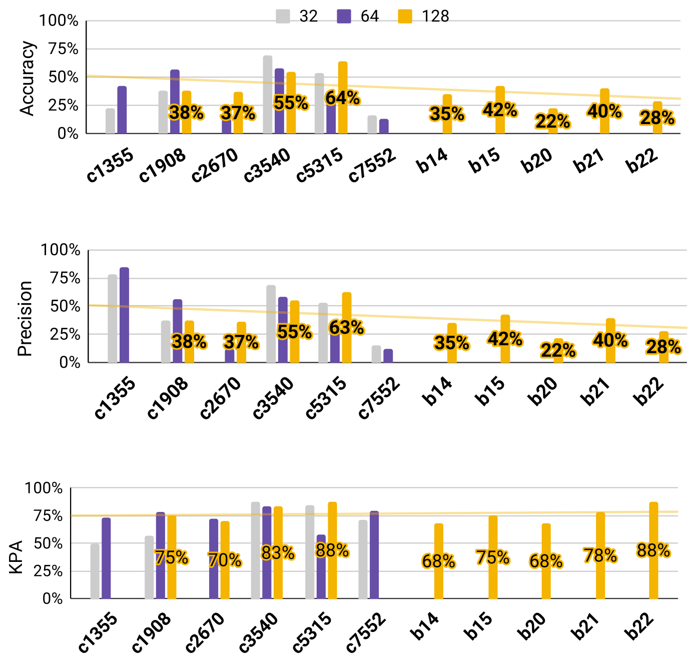

Figure 8: Results for the MuxLink attack on IsoLock Variant I, with both configured for default hop-sizes $h' = h = 3$. MuxLink threshold $th = 0$. Embedded data points refer to key-size of 128 bits.

scenarios, the attacks could not make any predictions, thus cannot misclassify any key-bit to begin with, resulting in an only theoretical PC value of 100%.

For layout evaluation, we report overheads after synthesis as outlined above, in %, over the unprotected baseline designs. We consider the following metrics: number of gates, load caps, area, and delay.

4.2 Security Evaluation

4.2.1 MuxLink [APSS22], Default Hop-Sizes

In Figures 8 and 9, we plot detailed results for the MuxLink attack on IsoLock Variant I, with both configured for default hop-sizes $h' = h = 3$. Results cover all achievable key-sizes for ISCAS-85 benchmarks but only the largest key-size for ITC-99 benchmarks; the latter is to demonstrate scalability.

First and foremost, we find that AC, a key metric for evaluating the resilience of locking schemes against structural ML attacks, is not surpassing random guessing (i.e., 50% AC) in almost all cases—IsoLock Variant I is effective.

Second, we find that all result are notably impacted by the classification threshold th. For $th = 1$, MuxLink makes significantly fewer predictions that are confirmed to be correct, i.e., there is a significant drop in AC when compared to $th = 0$. For $th = 0$, the AC values around 50% can be interpreted as approaching random guessing. This is simply because any prediction is made/allowed for this minimal threshold, irrespective of the predicted probability. At the same time, for $th = 1$, MuxLink achieves PC values of 100% in many cases; recall that this is a theoretical value in this scenario (Section 4.1.2). Both findings considered together imply that MuxLink can make only very few predictions for $th = 1$ to begin with, and it leaves more key-bits unresolved; this is expected as most predictions will remain below that maximal threshold.

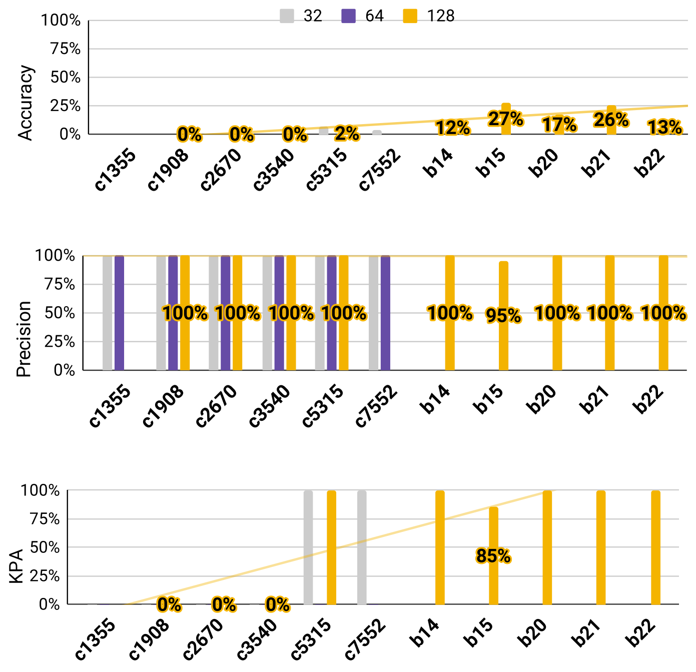

Figure 9: Results for the MuxLink attack on IsoLock Variant I, with both configured for default hop-sizes $h' = h = 3$. MuxLink threshold $th = 1$. Embedded data points refer to key-size of 128 bits.

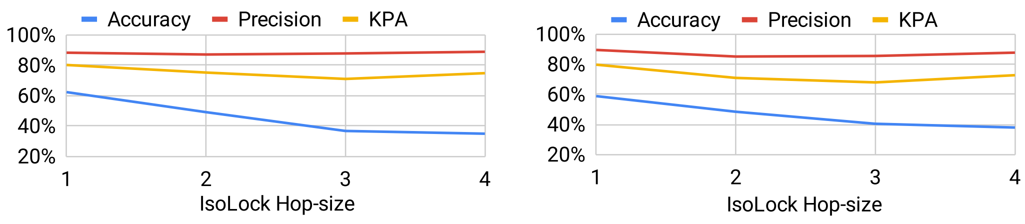

Figure 10: Average results for the MuxLink attack on IsoLock Variant I, considering all achieved key-sizes for each design. MuxLink is configured for $h' = 3$ (left) versus $h' = 4$ (right). For both scenarios, MuxLink threshold $th = 0$.

All this demonstrates that MuxLink is already strongly undermined by IsoLock Variant I, that is the variant with the most relaxed isomorphism check.

4.2.2 MuxLink [APSS22], Varying Hop-Sizes

In Figure 10, we summarize the impact of varying both hop-sizes h' for MuxLink and h for IsoLock Variant I.

First, the larger h is for IsoLock, the more challenged MuxLink is. The continuous downward trend for AC, reaching 50% and below for $h \geq 2$, indicates again that MuxLink cannot perform better than random guessing. Both PC and KPA remain relatively stable (i.e., for $th = 0$), confirming that MuxLink is not able to obtain more predictions, let alone more accurate ones, for larger h'.

Second, a related observation is that IsoLock should be implemented with h not much smaller than h'; otherwise, MuxLink can achieve upto 62% AC, i.e., for the observed best-case attack scenario with $h = 1, h' = 3$. This is much expected when isomorphism is enforced only at the smallest hop-size: MuxLink may well identify structural hints

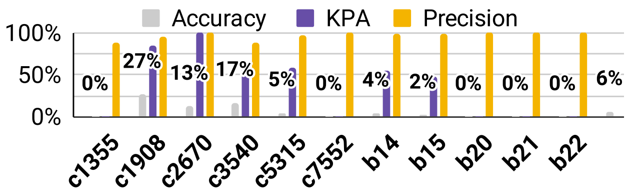

Figure 11: Average results for the SCOPE attack on IsoLock Variant I with $h = 3$, considering all achieved key-sizes for each design. Embedded data points refer to AC/accuracy.

around larger hop-sizes that can be learned on. Still, even for that best-case scenario – also considering this is for the most relaxed isomorphism check of IsoLock Variant I – MuxLink is far from inferring the full key.

Third, from the attack perspective, the impact of h' does not scale well. For larger h for IsoLock, the average AC achieved for larger $h' = 4$ is still following a strong and clear down-ward trend, with only marginal improvements of ≈ 3 percentage points (pp) over $h' = 3$. For smaller h, a larger $h' = 4$ is even counterproductive – recall that the general challenge for link-prediction attacks for large h' has been discussed in Section 3.1.

In short, MuxLink cannot perform much better than random guessing on IsoLock-ed design, even when scaling up the attack's hop-size and scaling down IsoLock's hop-size, all while considering only the relaxed isomorphism check of IsoLock Variant I. This is because the accuracy of MuxLink depends largely on the size of isomorphic key-gate structures, and the sizes achieved by our practical implementation are sufficient to prevent MuxLink from learning structural hints.

4.2.3 SAAM [SMRL21]

For all IsoLock-ed designs, SAAM returns 0% AC. This is because the attack principle of structural analysis naturally cannot infer any information in the presence of IsoLock's isomorphic key-gate structures. Based on this strong results, we refrain from reporting other metrics here.

4.2.4 SCOPE [ARB21]

In Figure 11, we show average results for the SCOPE attack. While precision is relatively high across all designs, recall that this metric optimistically assumes all undecided/unresolved key-bits as correct. KPA, and even more so AC, are more meaningful for judging IsoLock-ed designs – both metrics are approaching zero, especially for larger designs, which confirms the strong resilience of IsoLock also against SCOPE.

4.2.5 Redundancy [LO19]

In Figure 12, we show average results for the Redundancy attack. As with the other attacks, we note that PC remains high while KPA and AC are reaching toward zero. Thus, IsoLock also hinders the Redundancy attack.

4.2.6 Summary

We demonstrate that IsoLock is resilient against various oracle-less attacks, already in the most relaxed Variant I, forcing the accuracy of all attacks toward or below random guessing. This holds true across different benchmarks and different key-sizes. For best resilience, IsoLock should be implemented with sufficiently large hop-size, i.e., $h \geq 3$.

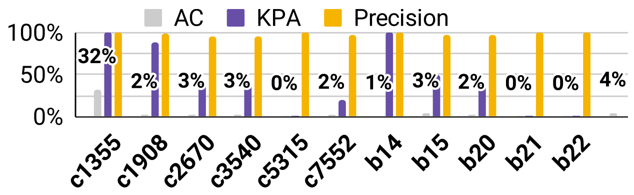

Figure 12: Average results for the Redundancy attack on IsoLock Variant I with $h = 3$, considering all achieved key-sizes for each design. Embedded data points refer to AC/accuracy.

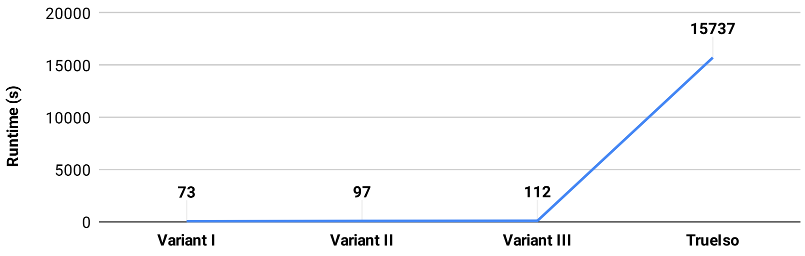

Figure 13: Average runtimes for IsoLock Variants I–III and TrueIso, all with $h = 3$, for key-size of 16 bits and considering ISCAS-85 benchmarks.

4.3 Runtimes for IsoLock

IsoLock Variant I incurs relatively short runtimes. More specifically, the average runtimes are around 69, 70, 65, and 51 minutes for hop-sizes h of 1, 2, 3, and 4, respectively, that is across all considered benchmarks and all achieved key-sizes.

For larger hop-sizes, runtimes tend to decrease, which indicates on favorable scalability of IsoLock Variant I. In general, searching for more isomorphic localities – required to achieve larger key-sizes – is inherently more challenging. Thus, for smaller hop-sizes, where smaller designs may still be locked with relatively large key-sizes arising from many, yet small, isomorphic localities, the average runtimes will be dictated by locking those smaller designs. For larger hop-sizes, however, smaller designs can, by construction, only be locked with fewer key-bits. Our practical implementation considers to short-cut the search in such cases, i.e., once the time taken for each locking step increases exponentially, the algorithm can be aborted. This helps to limit the average runtimes accordingly.

Runtimes for the different IsoLock variants are contrasted in Figure 13. For a fair comparison, we use the same, commonly achievable key-size of 16 bits here. We note that, for TrueIso, larger key-sizes of 32 bits or more are difficult to achieve for some ISCAS-85 designs and for all ITC-99 designs, i.e., at least for the considered time-out of 48h.

While all IsoLock variants exhibit similar and acceptable runtimes, IsoLock TrueIso requires considerable longer runtimes. This is expected, as TrueIso employs an exact, unrelaxed isomorphism check, w/o.l.o.g. based on the VF2 algorithm. That algorithm achieves the best performance for small-size graphs as well as larger sparse graphs; it can be limited for larger dense graphs [CFSV01, FSV01]. Next, we study the outlined trade-off in more detail.

4.4 Impact of Isomorphism Check

Here, we launch the MuxLink attack [APSS22] on all IsoLock variants and TrueIso, to study the impact of the isomorphism check underlying the locking algorithm on the efficacy against this state-of-the-art attack. We report the results achieved across ISCAS-85

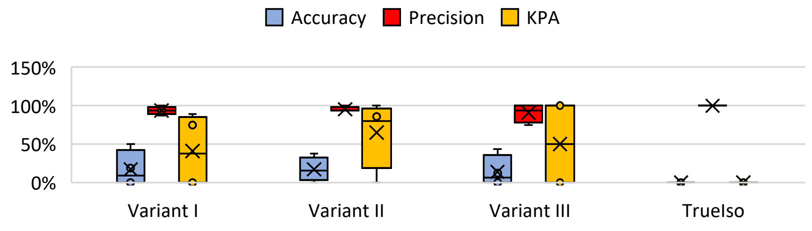

Figure 14: Results for the MuxLink attack on IsoLock Variants I–III and on TrueIso, for key-size of 16 bits, across ISCAS-85 benchmarks. Hop-sizes for IsoLock and MuxLink are $h = h' = 3$. MuxLink threshold $th = 0$.

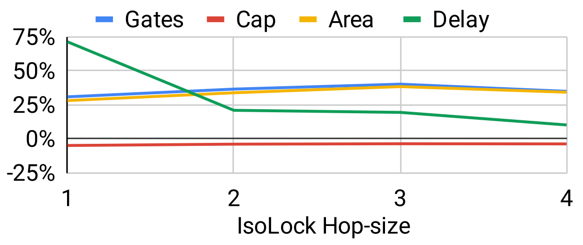

Figure 15: Average layout overheads for IsoLock Variant I, across all considered benchmarks and all achieved key-sizes.

benchmarks in Figure 14. Results for the three IsoLock variants differ only marginally, e.g., the average accuracy drops from 17% for Variant I to 14% for Variant III. For IsoLock TrueIso, results represent the worst-case scenario for the attack/best-case scenario for the defense: 0% AC and 0% KPA, along with an only-theoretical value of 100% PC.

In short, TrueIso achieves the best possible resilience against the MuxLink attack, whereas the three IsoLock variants achieve reasonable resilience as well. All in all, i.e., considering runtimes as well, we argue that the IsoLock variants provide practical trade-offs.

For future work, we may consider to improve the efficiency of TrueIso. However, given that isomorphism check is an NP-complete problem, options toward that end will be limited. Still, a meta-optimization approach, which selects from a number of different exact algorithms, individually considering the size, density, etc. of each netlist graph to be locked during selection and tuning of the algorithm, may be worthwhile to explore.

4.5 Layout Evaluation

In Fig 15, we summarize the overheads for IsoLock Variant I, across all designs and key-sizes, for varying hop-sizes h.

First, while average overheads are notable, it is important to emphasize that overheads amortize well for larger designs. For example, for the ITC-99 design b22_C for $h = 3$, we observe overheads of 3.81% for the number of gates, 0% for load caps, 4.86% for area, and 3.82% for delay, respectively. Overheads also tend to decrease for larger hop-sizes. Similar to runtimes discussed above, this is due to larger hop-sizes being more challenging to lock for smaller designs, leading to smaller average key-sizes and more limited layout overheads.

Second, when comparing to the D-MUX scheme, our overheads are competitive. For the default setting of $h = 3$, our average overheads are 40.22% for the number of gates, -3.64% for load caps, 38.42% for area, and 19.45% for delay, respectively. Note that D-MUX employs only 64 key-bits, whereas ours employs upto 128 key-bits, meaning that our overheads should be higher by construction. Still, in direct comparison,[6] our area overheads are -8.84 pp while our delay overheads are only +2.18 pp. While we do not explicitly evaluate power (due to limitations of *yosys-abc*), load caps can be considered as proxy metric for power: our average load-cap overheads of -3.64% indicate that there will be little power overheads, if any.

Third, we also observe similar trends for all other IsoLock variants; we refrain from reporting these results separately. These similar trends are due to the fact that the only difference between the variants is the isomorphism check, resulting in different gates selected/approved for locking, whereas the actual locking is implemented using the same MUX key-gate structures.

5 Conclusions

For the first time, we introduce the concept of *graph isomorphism* to the landscape of logic locking. We provide a provably-secure and learning-resilient scheme called *IsoLock*, founded on mathematical proofs. We also explore options to relax parts of the underlying theory, namely hop-sizes and search algorithms, both related to isomorphic localities. Accordingly, we are providing different relaxed variants of IsoLock for a practical implementation. We also like to emphasize that IsoLock is a generic concept: the core idea of inducing isomorphic key-gate structures via routing obfuscation can also be applied for other MUX-based locking strategies or even other key-gate structures.

Through our experimental study on ISCAS-85 and ITC-99 benchmarks, we show that runtimes are acceptable and layout overheads are competitive. The different variants enable good trade-offs for runtimes, achievable key-sizes and overheads, all while maintaining high learning resilience even for the most relaxed Variant I. More specifically, the state-of-the-art MuxLink attack can decipher on average only 5% of the key-bits for IsoLock-ed designs (Variant I), meaning that the attack fails on IsoLock. (This is while MuxLink has shown an accuracy of upto 100% on prior art, previously thought to be learning-resilient.) Under the SCOPE, SAAM, and Redundancy attacks, the average accuracy is 6%, 0%, and 4%, respectively, demonstrating well that IsoLock also fully protects against oracle-less attacks using approaches other than link-prediction and machine learning.

For future work, we seek to explore meta-optimization of different algorithms for exact isomorphism checks, and to extend IsoLock toward hindering oracle-guided attacks as well.

References

[AA03] Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3):211–230, 2003.

[AFB19] Abdulrahman Alaql, Domenic Forte, and Swarup Bhunia. Sweep to the secret: A constant propagation attack on logic locking. In *AsianHOST*, pages 1–6, 2019.

[AM13] Dmitry Skurt Abhijit Mahindroo, Nick Santhanam. The potential shake-up in semiconductor manufacturing business models, 2013.

[APH$^+$21a] Lilas Alrahis, Satwik Patnaik, Muhammad Abdullah Hanif, Hani Saleh, Muhammad Shafique, and Ozgur Sinanoglu. GNNUnlock+: A systematic methodology for designing graph neural networks-based oracle-less unlocking schemes for provably secure logic locking. *IEEE TETC*, pages 1–1, 2021.

[6]For D-MUX, reported area, delay, and power overheads are 47.26%, 17.27%, and 51.09%, respectively [SMRL21].

[APH$^+$21b] Lilas Alrahis, Satwik Patnaik, Muhammad Abdullah Hanif, Muhammad Shafique, and Ozgur Sinanoglu. UNTANGLE: unlocking routing and logic obfuscation using graph neural networks-based link prediction. In *ICCAD*, 2021.

[APK$^+$21] Lilas Alrahis, Satwik Patnaik, Johann Knechtel, Hani Saleh, Baker Mohammad, Mahmoud Al-Qutayri, and Ozgur Sinanoglu. UNSAIL: Thwarting oracle-less machine learning attacks on logic locking. *IEEE TIFS*, 16:2508–2523, 2021.

[APSS21] Lilas Alrahis, Satwik Patnaik, Muhammad Shafique, and Ozgur Sinanoglu. OMLA: An oracle-less machine learning-based attack on logic locking. *IEEE TCAS-II*, pages 1–1, 2021.

[APSS22] Lilas Alrahis, Satwik Patnaik, Muhammad Shafique, and Ozgur Sinanoglu. Muxlink: circumventing learning-resilient mux-locking using graph neural network-based link prediction. In *DATE*, pages 694–699, 2022.

[ARB21] Abdulrahman Alaql, Md Moshiur Rahman, and Swarup Bhunia. SCOPE: Synthesis-based constant propagation attack on logic locking. *IEEE TVLSI*, 29(8):1529–1542, 2021.

[CCAB21] Prabuddha Chakraborty, Jonathan Cruz, Abdulrahman Alaql, and Swarup Bhunia. SAIL: Analyzing structural artifacts of logic locking using machine learning. *IEEE TIFS*, 16:3828–3842, 2021.

[CCW18] Shuai Chen, Junlin Chen, and Lei Wang. A chip-level anti-reverse engineering technique. *JETC*, 14(2), jul 2018.

[CFSV01] Luigi Pietro Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. An improved algorithm for matching large graphs. In *3rd IAPR-TC15 workshop on graph-based representations in pattern recognition*, pages 149–159. Citeseer, 2001.

[FSV01] Pasquale Foggia, Carlo Sansone, and Mario Vento. A performance comparison of five algorithms for graph isomorphism. In *Proceedings of the 3rd IAPR TC-15 Workshop on Graph-based Representations in Pattern Recognition*, pages 188–199, 2001.

[HCS$^+$21] Wei Hu, Chip-Hong Chang, Anirban Sengupta, Swarup Bhunia, Ryan Kastner, and Hai Li. An overview of hardware security and trust: Threats, countermeasures, and design tools. *IEEE TCAD*, 40(6):1010–1038, 2021.

[KAHS20] Hadi Mardani Kamali, Kimia Zamiri Azar, Houman Homayoun, and Avesta Sasan. InterLock: An intercorrelated logic and routing locking. In *ICCAD*, pages 1–9, 2020.

[KW17] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[LHW18] Qimai Li, Zhichao Han, and Xiao Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, pages 3538–3545, 2018.

[LKK$^+$20] Nimisha Limaye, Emmanouil Kalligeros, Nikolaos Karousos, Irene G Karybali, and Ozgur Sinanoglu. Thwarting all logic locking attacks: Dishonest oracle with truly random logic locking. *IEEE TCAD*, 2020.

[LO19] Leon Li and Alex Orailoglu. Piercing logic locking keys through redundancy identification. In *DATE*, pages 540–545, 2019.

[PASK20] Satwik Patnaik, Mohammed Ashraf, Ozgur Sinanoglu, and Johann Knechtel. Obfuscating the interconnects: Low-cost and resilient full-chip layout camouflaging. *IEEE TCAD.*, 39(12):4466–4481, 2020.

[QBJKS06] Yanjun Qi, Ziv Bar-Joseph, and Judith Klein-Seetharaman. Evaluation of different biological data and computational classification methods for use in protein interaction prediction. *Proteins: Structure, Function, and Bioinformatics*, 63(3):490–500, 2006.

[RKM10] J.A. Roy, F. Koushanfar, and Igor L Markov. Ending Piracy of Integrated Circuits. *IEEE TC*, 43(10):30–38, 2010.

[RZZ$^+$15] J. Rajendran, Huan Zhang, Chi Zhang, G.S. Rose, Youngok Pino, O. Sinanoglu, and R. Karri. Fault Analysis-Based Logic Encryption. *IEEE TC*, 64(2):410–424, 2015.

[SMR$^+$21] Dominik Sisejkovic, Farhad Merchant, Lennart M. Reimann, Harshit Srivastava, Ahmed Hallawa, and Rainer Leupers. Challenging the security of logic locking schemes in the era of deep learning: A neuroevolutionary approach. *JETC*, 17(3), May 2021.

[SMRL21] Dominik Sisejkovic, Farhad Merchant, Lennart M Reimann, and Rainer Leupers. Deceptive logic locking for hardware integrity protection against machine learning attacks. *IEEE TCAD*, 2021.

[SRM15] Pramod Subramanyan, Sayak Ray, and Sharad Malik. Evaluating the Security of Logic Encryption Algorithms. In *HOST*, pages 137–143, 2015.

[SSC$^+$20] Akashdeep Saha, Sayandeep Saha, Siddhartha Chowdhury, Debdeep Mukhopadhyay, and Bhargab B Bhattacharya. LoPher: SAT-hardened logic embedding on block ciphers. In *DAC*, pages 1–6, 2020.

[W$^+$20] Clifford Wolf et al. Yosys open synthesis suite, 2012–20.

[Zaf] Ramish Zafar. TSMC's total 3nm investment will equal at least \$ 23 billion.

[Zaf22] Ramish Zafar. Tsmc's Advanced (3nm) Chip Production To Begin Next Month, Culling Delay Rumors - Report. https://wccftech.com/tsmcs-advanced-3nm-chip-production-to-begin-next-month-culling-delay-rumors-report/, Aug 2022. [Online; accessed 2022-09-22].

[ZC18] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *NIPS*, page 5171–5181, Red Hook, NY, USA, 2018. Curran Associates Inc.