Security Closure of IC Layouts Against Hardware Trojans

Fangzhou Wang[1], Qijing Wang[1], Bangqi Fu[1], Shui Jiang[1], Xiaopeng Zhang[1], Lilas Alrahis[2], Ozgur Sinanoglu[2], Johann Knechtel[2], Tsung-Yi Ho[1], and Evangeline F. Y. Young[1]
[1]The Chinese University of Hong Kong, [2]New York University Abu Dhabi
{fzwang,qjwang21,bqfu21,sjiang22,xpzhang}@cse.cuhk.edu.hk
{lma387,ozgursin,johann}@nyu.edu
{tyho,fyyoung}@cse.cuhk.edu.hk

ABSTRACT

Due to cost benefits, supply chains of integrated circuits (ICs) are largely outsourced nowadays. However, passing ICs through various third-party providers gives rise to many threats, like piracy of IC intellectual property or insertion of hardware Trojans, i.e., malicious circuit modifications.

In this work, we proactively and systematically harden the physical layouts of ICs against post-design insertion of Trojans. Toward that end, we propose a multiplexer-based logic-locking scheme that is (i) devised for layout-level Trojan prevention, (ii) resilient against state-of-the-art, oracle-less machine learning attacks, and (iii) fully integrated into a tailored, yet generic, commercial-grade design flow. Our work provides in-depth security and layout analysis on a challenging benchmark suite. We show that ours can render layouts resilient, with reasonable overheads, against Trojan insertion in general and also against second-order attacks (i.e., adversaries seeking to bypass the locking defense in an oracle-less setting).

We release our layout artifacts for independent verification [29] and we will release our methodology's source code.

KEYWORDS

Hardware Trojans, physical design, security closure, logic locking, ISPD'22 contest

1 INTRODUCTION

In this work, we focus on the threat of hardware Trojans, i.e., malicious circuit modifications. By design, Trojans are only minor in extent but severe in fallout [9, 31, 32]. Many Trojan countermeasures have been proposed over the years (see also Sec. 2.1.2 and 3). Aside from (i) reactive, post-silicon monitoring and (ii) pre-/post-silicon verification, testing, detection, and inspection, we argue that (iii) proactive, pre-silicon prevention is essential to hinder Trojans to begin with. However, most related prior art falls short in terms of resilience against advanced attacks and/or overheads.

The **objective** of this work is security closure against hardware Trojans. Note that security closure is an emerging paradigm to proactively harden the physical layouts of integrated circuits (ICs) at design-time against various threats that are executed post-design time [17, 18]. In this work, we aim for *proactive, pre-silicon Trojan prevention, by carefully and systematically hardening the physical layout of ICs as a whole against post-design Trojan insertion.*

ISPD 2023, March 26–29, 2023, Online

Toward that end, and more so for Trojan defense in general, we have identified the following **research challenges**:

- *Robustness – Any defense must remain robust in place.* That is, a foundry-based adversary, being fully aware that some Trojan defense is put in place, would naturally want to first circumvent that defense (i.e., stealthily remove, override, or otherwise render useless) before inserting their Trojan. Such second-order attacks represent a key challenge where prior art, be they proactive or reactive defenses, falls short.
- *Effectiveness – Any defense should be able to protect against various Trojans.* This is especially true for proactive, pre-silicon schemes which have only "one shot" at design-time. Layouts should be protected as a whole, i.e., in terms of (i) layout resources that are exploited for Trojan insertion (e.g., open placement sites, free routing tracks, available timing slacks) and (ii) structure and functionality, analysed by adversaries for targeted Trojan insertion.
- *Efficiency – Any defense should incur limited, controllable overheads.* Taking control over such trade-offs requires two parts: (i) metrics for security and overheads, and (ii) some integrated, secure-by-design methodology, utilizing the metrics and user guidance as needed.

Accordingly, our work makes the following **contributions**:

(1) Layout-level logic locking – We propose a multiplexer (MUX)-based locking scheme, called *TroMUX*, devised to hinder post-design Trojan insertion. Our locking scheme addresses the above challenges through the following capabilities:
   (a) Robustness – *TroMUX* is devised to withstand state-of-the-art, machine learning (ML)-based attacks on locking, like *SCOPE* [1] and *MuxLink* [2]. This is essential to hinder adversaries from circumventing the Trojan defense.
   (b) Effectiveness – By the security promise of locking, attackers cannot easily insert targeted Trojans anymore, i.e., Trojans that require understanding of the original design. Furthermore, by densely filling up the layout with as many *TroMUX* instances as practically possible (keeping the design quality well under control), attackers cannot easily insert additional Trojan logic in general anymore.
(2) Integrated security closure – We propose an effective and efficient methodology for security closure against post-design Trojan insertion. The methodology is fully integrated into a commercial-grade physical-synthesis flow. Such integration is essential to achieve (i) the above outlined security principles and (ii) take control of security-versus-overheads trade-offs arising for security closure of layouts.

Release: We will release our source codes. We release our artifacts of secured layouts already during pre-publication [29], to enable independent verification of our work.

2 BACKGROUND AND MOTIVATION

2.1 Trojans

2.1.1 Working Principles. Trojans are malicious hardware modifications [9, 31]. This notion is very diverse, covering modifications that: (i) leak information from an IC, reduce its performance, or disrupt its working altogether; (ii) are always on, triggered internally, or triggered externally; (iii) are introduced through untrustworthy third-party intellectual property (IP), adversarial designers, during outsourced mask generation, manufacturing, packaging of ICs; etc.

Most if not all Trojans comprise a trigger and a payload; the trigger activates the payload on some attack condition, and the payload serves to perform an actual attack. Triggers are often based on low-controllability nets (LCNs)—to complicate their detection during testing—whereas payloads are targeting on sensitive assets like key registers. Note that trigger and payload are implemented individually but are working together in tandem. Also note that most Trojans require some layout-level resources like open placement sites, free routing tracks, and/or available timing slacks.

2.1.2 Prior Art for Countermeasures. Prior art for Trojan countermeasures can be classified into:

- Proactive detection schemes, that is pre-silicon verification (e.g., [7, 9]), post-silicon testing (e.g., [8, 9]), or post-silicon inspection (e.g., [25]);
- Proactive, pre-silicon prevention schemes (e.g., [4, 5, 10, 13, 17, 19, 22, 27, 30]);
- Reactive, post-silicon monitoring schemes (e.g., [11, 14, 26]).

Proactive detection and reactive monitoring schemes are generally challenged by advanced and stealthy Trojans [12, 32]. Further, reactive monitoring as well as proactive prevention schemes typically require some dedicated hardware support—if not secured properly, the related circuitry may well be circumvented by adversaries during the course of Trojan insertion.

Given these challenges, a *robust* scheme for proactive prevention is essential to hinder Trojans early on, as best as possible. We discuss prior art for proactive, pre-silicon prevention in more detail in Sec. 3. A comparison of prior art and ours is also outlined in Table 1.

Finally, note that the three classes are orthogonal, yet generally compatible. To increase the overall resilience against Trojans, techniques from all classes could (and should) be utilized jointly, if permissible in terms of design overheads and cost.

2.2 Security Closure

As indicated, security closure is an emerging paradigm that seeks to proactively harden the physical layouts of ICs, at design time, against various threats that are executed post-design time [17, 18]. In the broader context of secure-by-design efforts for electronic design automation (EDA) tools [15, 21, 23], security closure aims for secure physical-synthesis stages.

Security closure against Trojans means to control physical synthesis such that insertion of Trojans becomes impractical, while at the same time managing the impact on design quality of such measures [17, 18]. For example, an aggressively dense layout would leave only few open placement sites and few routing resources exploitable for Trojan insertion. While aggressively dense layouts are already challenging by themselves, in terms of managing design quality, such naive approach is still not good enough for security closure. This is because, for one, an advanced Trojan like *A2* [32] may require only 20 placement sites for an advanced analog implementation [24];[1] such very few sites are likely to remain even in aggressively dense layouts. For another, imagine a second-order attack where an adversary would first revise the layout as much as necessary (but also as little as possible, to avoid subsequent detection), and only then insert their Trojan.

In short, successful efforts for security closure against Trojans need to holistically address the challenges outlined in Sec. 1.

Table 1: High-Level Comparison with Prior Art

Work	Approach	Robust	Effective[a] Targ. / Untarg.	Efficient	Artifacts Available
[10]	Locking	N [10, 22]	N$^{a'}$ / N	(Y)	N
[19, 22]	Locking	(N)[b]	(N)$^{a'}$ / N	(Y)	N
[27]	Locking	(?)	(?)$^{a'}$ / N	(Y)	N
[4, 5, 30]	Addit. Logic	(?)	N / (?)$^{a'}$	(Y)	N
[13, 17]	Phys. Synth.	(N)[c]	N / (N)$^{a'}$	Y	N
Ours: *TroMUX*	**Locking and Layout Filling**	**Y**	**Y / (Y)**	**Y**	**Y [29]**

Notation: Y – yes, N – no, (?) – unclear, (Y) – yes but some caveat, (N) – unlikely
[a] Two scenarios are differentiated: effective against targeted Trojans / against untargeted Trojans. Robustness impacts effectiveness; related cases are labelled via a'.
[b] The employed locking scheme has been broken in [3].
[c] We argue that such schemes can be reverted by adversaries.

2.3 Logic Locking

Logic locking, or locking for short, means to incorporate so-called key-gate structures that are controlled by secret key-bits. While locking is largely known for protection of IC's IP, it can also serve for Trojan defense. In fact, different such schemes have been proposed: AND/OR locking [10], X(N)OR locking [19, 22], and custom locking [27]. We discuss relevant prior art in more detail in Sec. 3.1.

Recently, ML-based attacks like *SAIL* [6], *SCOPE* [1], *OMLA* [3] and *MuxLink* [2] succeeded in learning various design features and subsequently predicting key-gates, all in an oracle-less setting,[2] i.e., without need for a functional chip that can be queried for its functional behaviour. For example, *MuxLink* learns on the structure of regular, unlocked parts of the design at hand, using a graph representation of the design, and then deciphers the MUX key-gates. More specifically, *MuxLink* considers each MUX key-gate's input at a time, connecting it to the output of the MUX, thereby essentially bypassing the key-gate, and contrasts the resulting different structures to the trained knowledge to predict the more likely structure.

[1] This is a remarkable exception—other Trojans reported in the literature, as well as a digital version of *A2* itself, require hundreds or thousands of sites [24].
[2] Only the oracle-less setting is relevant for our work as we are utilizing locking for proactive, pre-silicon prevention of Trojans, not for IP protection against end-users.

In short, ML-resilient locking schemes remain an open challenge, but are essential for locking-based, proactive Trojan prevention.

3 PRIOR ART

3.1 Locking-Based Trojan Prevention

Dupuis et al. [10] lock LCNs using AND/OR key-gates, to hinder insertion of Trojan triggers. They consider timing slacks, varying toggling thresholds, and balanced switching probabilities. Samimi et al. [22] follow similar principles as Dupuis et al., but utilize X(N)OR key-gates. Marcelli et al. [19] propose a multi-objective algorithm, seeking to minimize LCNs and maximize the efficacy of X(N)OR locking at the same time. Šišejković et al. [27] secure inter-module control signals against software-controlled hardware Trojans, using encryption circuitry along with regular locking. For the latter, they do not specify/limit the type of key-gates.

The above prior art has limitations as follows.

- Dupuis et al. [10] cannot protect against insertion of targeted Trojans. This is because they utilize AND/OR key-gates to tune the controllability of nets, not to obfuscate the design.
- Šišejković et al. [27] only protect against Trojans targeting on inter-module control signals, not on modules' internals.
- None consider Trojan prevention at the layout level; all are working only at netlist level. This implies that none can conclusively prevent insertion of Trojan logic into the layout.
- None explicitly prevent Trojan payloads. Šišejković et al. [27] are using locking in general, without focus on triggers or payloads, and others are locking only low-controllability nets (to hinder insertion of Trojan triggers).
- None show robustness against ML-based attacks; all are prone to second-order attacks bypassing the locking defense. This is most evident for Dupuis et al. [10]: they employ a direct, hard-coded correlation of key-bit '0' to OR key-gates, key-bit '1' to AND key-gates, respectively, and do not employ re-synthesis, which is essential for obfuscation of that correlation, but would be challenging for their objective of tuning the controllability of nets.

Note that, in contrast, our work addresses all these limitations.

3.2 Layout-Level Trojan Prevention

Xiao et al. [30] fill layouts with built-in self-test components, arguing that tampering of those structures (by adversaries regaining layout resources to insert their Trojans) is detected by post-silicon testing. Similarly, Ba et al. [4, 5] fill layouts with additional circuitry. Knechtel et al. [17] propose techniques for Trojan-resistant physical synthesis, based on locally increasing placement density (to hinder Trojan insertion) and locally increasing routing density (to hinder Trojan routing). Hossein-Talaee et al. [13] redistribute white space/open sites otherwise exploitable for Trojan insertion.

The above prior art has limitations as follows.

- None protects against insertion of targeted Trojans. In the absence of locking or other obfuscation schemes, the original design is fully accessible by adversaries.
- None conclusively shows robustness against second-order attacks, as in adversaries regaining layout resources despite the defense put in place. For example, for the work in [13], shifting of white spaces can be trivially reverted.
- For full efficacy, the work in [30] requires 100% test coverage which can be difficult to achieve. Further, the methodology is challenged by high utilization rates, limiting its practicability.
- For the work in [4, 5, 30], the number of additionally required primary inputs scales with layout filling. This is impractical; pads for primary inputs/outputs (PIs/POs) are large in actual ICs and, if not employed wisely, can considerably increase the chip outline, directly increasing cost for silicon area.

Note that, in contrast, our work addresses all these limitations.

4 THREAT MODELS

Trojans: We follow a classical threat model that is consistent with the literature as follows.

(1) We assume adversaries reside within manufacturing facilities, whereas the design process is considered trustworthy.
(2) From (1) follows that adversaries have no knowledge of the original unprotected design, only of the protected physical layout at hand.
(3) From (1) also follows that we do not account for Trojans introduced by, e.g., malicious designers or third-party IP modules. Against such threats, and to increase a layout's overall Trojan resilience, countermeasures orthogonal to ours, namely proactive detection or reactive monitoring schemes (Sec. 2.1.2), may be applied along with ours.
(4) We assume Trojans are implemented using regular standard cells, not by modifying interconnects or transistors, etc.
(5) We assume targeted Trojans with triggers based on LCNs and payloads targeting on assets, i.e., security-critical components like key registers.
(6) The adversaries' objective is to insert some Trojan(s), more specifically trigger and payload components, into the layout.

Locking: First, recall that we employ locking not for IP protection, but to hinder Trojan insertion. Toward that end, we follow an oracle-less threat model that is consistent with both the literature and the above threat model on Trojans as follows.

(1) The adversaries' objective is to circumvent the locking scheme, to (a) understand the true functionality of the design, required for targeted Trojan insertion, and (b) to regain layout resources, required for Trojan insertion in general.
(2) Given that adversaries are assumed to reside within manufacturing facilities, only oracle-less attacks are applicable. None of the well-known Boolean satisfiability-based attacks are applicable. Neither applicable are oracle-less attacks on locking schemes unlike *TroMUX*, e.g., [33].
(3) Following Kerckhoffs' principle, all implementation details for *TroMUX* locking are known to the adversaries; only the key-bits remain undisclosed.

5 METHODOLOGY

This work is motivated by the need for a proactive, pre-silicon Trojan-prevention scheme that is robust, effective, and efficient. As discussed in Sec. 3, prior art falls short toward that end.

Our approach—which can be summarized as locking and layout filling applied in unison—aims to hinder first-order and second-order Trojan-insertion attacks. We integrate an ML-resilient, MUX-based locking scheme directly into the IC layouts. Unlike prior art, we neither employ trivial filler/spare cells, nor separate circuitry, nor vulnerable locking schemes. To protect IC layouts holistically, our methodology carefully embeds, in a security-aware manner, as many locking instances during physical instances as practically possible, i.e., keeping the design quality well under control. Our methodology is fully integrated into commercial-grade design tools.

Next, we describe the components of our methodology, i.e., a MUX-based locking scheme and a physical-synthesis flow for security- and design-aware locking and layout filling. Note that both are devised to be working in unison, but also require individual solutions toward that end.

5.1 *TroMUX*

Our MUX-based locking scheme, *TroMUX*, is specifically devised to hinder post-design Trojan insertion. Given the rise of powerful ML-based attacks on simple locking schemes, e.g., on X(N)OR key-gates [3], we opt for MUX-based locking which is considered more resilient [28]. Still, even MUX-based schemes have been attacked recently [1, 2]. To render *TroMUX* resilient against such advanced ML-based attacks, we devise the following implementation.

Locking Approach. Unlike prior art, *TroMUX* is *not* based on obfuscating the netlist connectivity through MUX key-gates, but on using MUX key-gates for regular locking. Thus, for *TroMUX*, there is no information leakage arising from MUX key-gates for the connectivity and structure of the design, which is the key vulnerability of prior art [2]. Instead, *TroMUX* employs simple but resilient key-gate structures with localized connectivity, described next.

Learning-Resilient Key-Gate Structures. The design of *TroMUX* ensures that key-bits are fully randomized, without any correlation to each of the locked gate's true functionality. To do so, *TroMUX* instances render the locked gates interchangeable with respect to their complementary counterparts, e.g., a NAND gate can act as AND or NAND, only depending on the key-bit.

When locking gates using *TroMUX* instances, one picks randomly from the different possible configurations (Fig. 1). It is essential to note how the different configurations are structurally indistinguishable; only the key-bits determine the true functionality. Also, key-bits are easy to randomize for any particular true functionality, simply by randomly picking one of the possible configurations. For simple complementary gates, like (a) AND and (b) NAND, there are four different pairwise configurations which are indistinguishable on their own. Note how half the configurations are based on first transforming the original gates to their counterparts; this serves for further obfuscation of the overall layout regarding the distribution of gate types.

Locking of Complex Gates. Concerning flip-flops (FFs), these can be locked as is, by connecting the Q and QN ports to a dedicated key-gate structure (Fig. 1(c)).[3] For commercial libraries, where FF outputs are typically optimized and trimmed, similar key-gate structures as for simple gates can be initiated (Fig. 1(d)).

[3]We note that, independently of our work, this key-gate structure was proposed by Karmakar et al. [16], though in the context of locking scan chains.

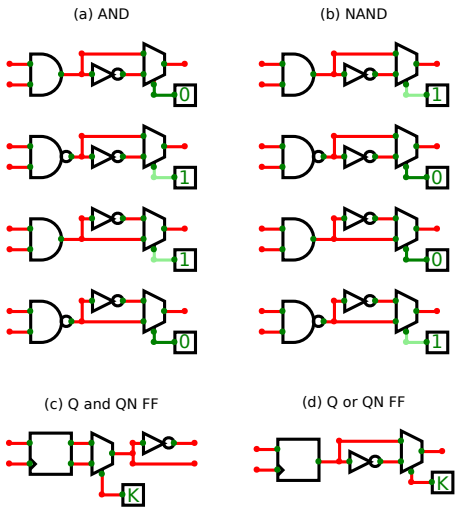

Figure 1: Design of different *TroMUX* instances. The key-bit is connected to the MUX select line. (a, b) Locking of complementary (N)AND gates; other pairings of simple gates are locked similarly. (c, d) Locking of flip-flops (FFs) with different output configurations.

For other complex gates like AOI, we observe that complementary counterparts appear rarely in an optimized layout, if they are available at all in the library. Thus, we defer locking of such complex gates as follows. When such complex gate is selected for locking, we search its fan-out in depth-first manner for simple gates and FFs, covering all the fan-out paths. Then, all the found simple gates and FFs are locked instead, which essentially translates to locking the downstream structure of that complex gate.

No Information Leakage from Physical Layout. As indicated, there are *no* direct correlations between the types of original versus the locked gate, the connectivity within and across *TroMUX* instances, and the correct key-bit; all are interchangeable, randomly chosen, and thus indistinguishable for an attacker.

It is also important to note that the two internal *TroMUX* nets connecting to the MUX inputs (or driving the MUX fan-out for the case of Q and QN FFs, Fig. 1(c)) share a path and are, thus, both optimized/impacted at once by EDA tools. Accordingly, an attacker seeking to study the underlying timing paths, driver strengths, etc., would not gain any additional information.

5.2 Physical Synthesis

With the proposed locking scheme, we introduce a physical-synthesis flow that is capable of hardening any post-route layout with negligible timing and area overheads. As outlined in Fig. 2, we start with the original netlist and other necessary data to generate a baseline layout and then carefully interleave logic locking and physical synthesis to produce a final protected layout.

5.2.1 Two-Stage Locking Scheme. In the first stage, we lock all security-critical FFs, as defined by assets. After locking, we conduct placement and routing (P&R), and obtain a partially protected layout (PPL). A PPL has only FF assets protected; no other parts like

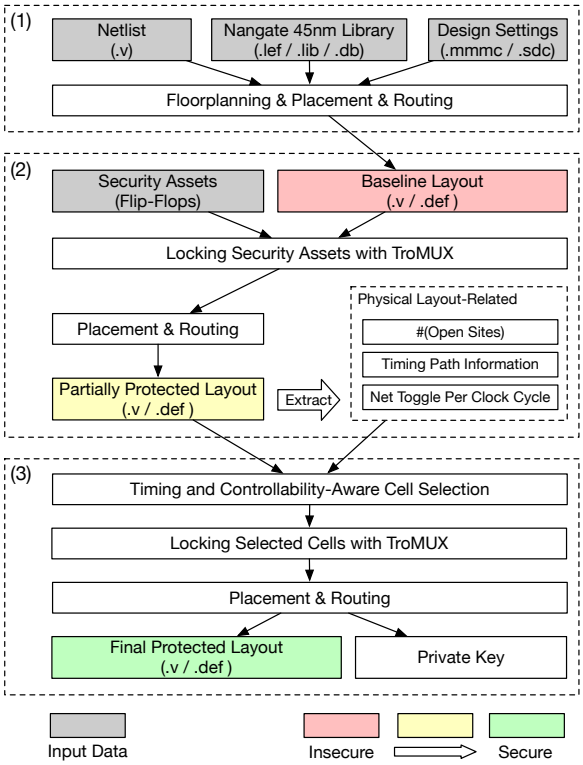

Figure 2: Our physical-synthesis flow, which consists of three parts: (1) initial synthesis, (2) locking of security assets, and (3) locking considering timing and controllability.

LCNs and LCCs (i.e., the cells driving LCNs; both are targets for Trojan triggers) are protected yet. Further, a PPL has relatively low utilization, leaving placement sites exposed to Trojan insertion.

In the second stage, we thus aim at locking as many cells as possible to fill the layout, also prioritizing the locking of LCCs, all without degrading the design quality. With the physical layout-related information obtained from the PPL, we first derive the number of cells to be locked, then perform cell selection considering timing and controllability (detailed in Sec. 5.2.4). After locking all selected cells, P&R is re-run, and we finally retrieve a highly-utilized layout with all security-critical components well protected through both *TroMUX* locking and much fewer placement sites left open.

5.2.2 Storage of Key-Bits. For each *TroMUX* instance, there will be one corresponding key-bit, requiring some facility to store all key-bits. We employ a large shift register for the following reasons:

- Prior art often considers adding a dedicated PI for each key-bit, which is not scalable. For us, we only require two additional PIs (data in, load) for any number of key-bits.
- Ours incurs negligible impact on performance and power,[4] while the impact on area is even desired (see below).

[4]For a design locked with k key-bits, loading will take k clock cycles. This is done during initial boot-up, when the main circuitry is still hold in reset; runtime cost for such one-time initialization are considered negligible. Once the load signal signal is set low, the key-bits will remain stable, consuming only some static FF power.

Table 2: Notations for Cell Selection

Term	Description
C	The set of standard cell instances
c	A cell instance from C
N	The set of nets
n	A net from N
$N(c)$	The set of nets driven by c
P	The set of timing paths
$P(n)$	The set of timing paths covering n
$MS(n,P)$	The minimum slack of paths covering n
$TPC(n)$	The number of toggles per clock cycle for n

- The FFs used to build up the shift register are also helpful for locally filling open placement sites as needed, whereas bulky memory blocks would rather complicate this task.
- Finally, memory blocks are unavailable for the library used in this work, i.e., *Nangate 45nm Open Cell Library* [20]. We require this library for the recently introduced ISPD'22 benchmarks for security closure of physical layouts [18].

5.2.3 On-Demand Key Length. *TroMUX* instances occupy open placement sites with their INVs, MUXes, and FFs. After locking security assets, we determine the additional key length required to fill the physical layout at hand as follows:

$$k = floor\left(\frac{num\_open\_sites}{size(INV) + size(MUX) + size(FF) + \alpha}\right) \quad (1)$$

where $size(INV)$ represents the size of the smallest INV cell in the library, etc., and α is a parameter for timing budget. As commercial tools will usually conduct timing optimization by gate sizing and buffer/inverter insertion, we need to reserve some additional space during locking. Accordingly, for designs where timing closure is more challenging, we will need a larger α, and vice versa.

5.2.4 Cell Selection Considering Timing and Controllability. Recall that, in the second locking stage, we lock selected cells to reduce the number of open sites and protect more LCNs/LCCs. As key-gate structures can introduce further cell delays to related timing paths, the selection of cells to lock becomes critical for timing closure. Thus, we propose a scoring function $cellScore(c, N, P)$ to comprehensively describe the priority of a cell c as follows.

$$cellScore(c, N, P) = \sum_{n \in N(c)} netScore(n, P), \quad (2)$$

$$netScore(n, P) = \frac{1}{1 + exp(-2 \cdot MS(n, P))} \cdot \frac{1}{TPC(n) + 10^{-3}}, \quad (3)$$

$$MS(n, P) = \begin{cases} \min\limits_{p \in P(n)} p.slack & , \text{if } |P(n)| > 0, \\ -0.5 & , \text{otherwise}, \end{cases} \quad (4)$$

with terms described in Table 2. The cell score is represented as the sum of net scores to generalize to multi-output cases, e.g., both Q and QN of a FF are used. Further, we devise $netScore(n, P)$ to jointly consider timing and controllability. Using sigmoid, the score value remains positive even for negative but small slacks. Besides, since $TPC(n) \in [0, 2]$ and nets with $TPC \leq 0.1$ are considered as

LCNs in this work, we add a small margin (10^{-3}) to avoid both div-by-0 and $TPC(n)$ from dominating the score. When calculating MS, some nets may not be covered by any reported timing paths (due to limitations of the commercial tool). For those nets, $MS(n, P)$ returns -0.5, a fall-back value close to average negative slacks observed; this value serves to conservatively penalize cells with unknown timing.

Algorithm 1 Cell Selection Considering Timing and Controllability

Input: Standard Cells C, Nets N, Timing Paths P, Key Length K, Locking Delay σ.
Output: The Set of Cells to Lock C'.

```
1:  C' ← {};
2:  while |C'| < K do
3:      foreach cell c ∈ C do
4:          c.score ← cellScore(c, N, P);        ▷ Score calculation for each cell
5:      Let c_h be the cell with the highest score in C;
6:      C'.append(c_h);
7:      C.remove(c_h);
8:      foreach net n ∈ N(c_h) do
9:          foreach timing path p ∈ P(n) do
10:             p.slack ← p.slack − σ;           ▷ Pessimistic slack estimation
11: return C';
```

Using the above scoring, we propose an iterative cell-selection algorithm in Algorithm 1. In each iteration, we calculate cell scores (line 3–4) and pick those cells with the highest score (line 5–7). Next, we update the slacks of affected timing paths based on σ, a pessimistic estimate of delay introduced by locking, defined as the sum of worst-case delays for INV_X1 and MUX2_X1 (i.e., the default *TroMUX* cells for our experiments), as derived from the libraries for the matching corner cases (line 8–10).

Our initial experiments show that the proposed two-stage locking scheme is more timing-friendly over a single-stage approach. That is, when we did directly use the physical information extracted from the baseline layouts (marked in red in Fig. 2) for cell selection and related locking, it was much harder to achieve timing closure, due to the accumulation of slack estimation errors.

6 EXPERIMENTAL RESULTS

6.1 Setup

6.1.1 Tools. As indicated, we base our work on a commercial-grade design flow. Without loss of generality, we use *Cadence Innovus 20.14* for physical synthesis. The methodology is implemented in custom *TCL* scripts and *Python* code. For security analysis using *SCOPE* and *MuxLink*, setup details are provided in Sec. 6.3.

6.1.2 Benchmarks. We employ the benchmark suite from the ISPD'22 contest on security closure [18]. The suite comprises a range of crypto cores as well as the *openMSP430* microcontroller. As Table 3 shows, the designs vary in terms of complexity, utilization, size (cells, nets), available metal layers, timing constraints, and corners. Since the suite was synthesized by legacy versions of *Cadence Innovus*, we resynthesize all designs at our end with floorplan utilization rates similar to the original post-route benchmark layouts; only the rates for *AES_3*, *openMSP430_2*, and *TDEA* are set 10% lower, as needed to lock all security assets.

6.2 Results on the ISPD'22 Benchmark Suite

We quantify security and layout results for the resynthesized baseline layouts versus the final, protected layouts in Table 4.

Table 3: Benchmark Statistics

Design	F. Utils	#(Cells)	#(Nets)	#(ML)	CP	Corner
AES_1	75.0%	16509	19694	10	1	typical
AES_2	75.0%	16509	19694	10	1	typical
AES_3	85.0%	15836	19020	10	1	typical
Camellia	50.0%	6710	7160	6	10	slow
CAST	50.0%	12682	13057	6	10	slow
MISTY	50.0%	9517	9904	6	10	slow
openMSP430_1	50.0%	4690	5312	6	30	slow
openMSP430_2	70.0%	5921	6550	6	8	slow
PRESENT	50.0%	868	1046	6	10	slow
SEED	50.0%	12682	13057	6	10	slow
SPARX	50.0%	8146	10884	6	10	slow
TDEA	70.0%	2269	2594	6	4	slow

* F. Utils: floorplanning utilization for resynthesis; #(Cells/Nets): number of cells/nets in original netlists; #(ML): number of metal layers; CP (ns): clock period; Corner: typical is characterized for 1.1V and 25C, slow for 0.95V and 125C.

First, all the protected layouts show much higher utilization rates, thereby reducing open placement sites by 90.3% on average and rendering designs much more resilient against Trojan insertion in general. Meanwhile, we achieve higher track utilization (defined as *total routed wire length* / *total track length*) compared with the baseline layout such that routing from triggers to payloads will be more challenging for the attackers. Second, as another layer of protection, recall that all security assets are locked and that LCNs/LCCs are well locked in most layouts, except for those with relatively high initial utilizations. Third, all layouts are without any design rule check (DRC) violations, despite the ultra-high utilization. This demonstrates the effectiveness of our proposed flow. Fourth, total power is increased by 18.5% on average, which seems reasonable given all the additional cells introduced with *TroMUX* instances.

In Fig. 3, we show the layouts of the exemplary *Camellia* design in three stages as described in Fig. 2. After locking assets, there are still a number of open sites. However, with the second locking stage, we manage to increase the utilization to as high as 98.9%.

6.3 ML-Based Attack Analysis

Recall that we use logic locking to both protect security-critical components in particular and the layout in general. Attackers would want to undermine our locking and remove *TroMUX* instances, to be able to insert Trojans targeted on assets and LCNs/LCCs in particular (Sec. 4). Thus, we evaluate ours against state-of-the-art ML-based attacks *SCOPE* [1] and *MuxLink* [2].

Setup for SCOPE. The attack expects designs in *bench* format. Thus, we resynthesize our locked benchmarks using the *bench*-specific, limited set of cells, (i.e., AND, OR, NAND, NOR, INV, BUF, DFF, XOR, and XNOR) and convert the netlists into *bench* format using an in-house *Python* script. Since the attack cannot handle loops in the locked designs, we represent FFs as pseudo PIs/POs. Further, we use the default margin value of 0 [1].

Setup for MuxLink. The original implementation supports only selected gates (i.e., AND, NAND, OR, NOR, INV, XOR, XNOR, BUF). Since we use the ISPD'22 benchmarks where all gates from the *Nangate* library are used, we extend the one-hot feature vectors of *MuxLink* accordingly. Consistent with the original operation of

Table 4: Layout and Security Results for Ours on the ISPD'22 Contest Benchmark Suite

Design	Baseline Layout (Resynthesized)						Protected Layout (Final)									
	Utils	#(Open)	TU	WNS	TNS	Power	Utils	#(Open)	Δ(Open)	TU	WNS	TNS	Power	#(LSA)/#(SA)	#(LLCC)/#(LCC)	KL
AES_1	75.4%	43,980	8.7%	0.000	0.000	59.957	96.2%	6,838	-84.5%	11.1%	-0.013	-0.043	60.552	291/291	884/1,389	1,199
AES_2	75.1%	44,420	8.7%	-0.001	-0.003	59.441	96.3%	6,649	-85.0%	10.9%	-0.008	-0.017	61.040	291/291	908/1,431	1,202
AES_3	85.7%	22,129	9.7%	-0.001	-0.002	59.869	96.6%	5,225	-76.4%	11.2%	-0.031	-4.657	62.787	291/291	150/1,310	441
Camellia	49.5%	33,919	9.5%	1.194	0.000	1.233	98.9%	753	-97.8%	13.3%	0.015	0.000	1.488	256/256	724/724	1,271
CAST	49.1%	54,444	9.1%	0.047	0.000	3.136	93.6%	6,879	-87.4%	12.7%	-0.134	-1.455	3.912	192/192	983/994	1,572
MISTY	48.5%	43,359	8.6%	-0.021	-0.037	2.238	94.4%	4,686	-89.2%	12.4%	-0.140	-1.515	2.966	204/204	307/312	1,215
openMSP430_1	49.7%	32,799	6.2%	0.000	0.000	0.375	97.9%	1,389	-95.8%	12.6%	0.000	0.000	0.544	340/340	441/456	1,218
openMSP430_2	70.5%	15,125	7.7%	0.000	0.000	1.239	97.1%	1,497	-90.1%	10.6%	-0.006	-0.015	1.369	334/334	209/696	543
PRESENT	49.8%	6,284	4.4%	6.694	0.000	0.198	98.0%	245	-96.1%	6.8%	4.935	0.000	0.235	80/80	3/3	241
SEED	49.1%	54,444	9.1%	0.047	0.000	3.136	94.3%	6,056	-88.9%	12.7%	-0.182	-1.072	3.908	195/195	979/989	1,612
SPARX	49.9%	69,979	6.3%	2.452	0.000	2.164	98.8%	1,658	-97.6%	14.0%	0.027	0.000	2.822	2,176/2,176	361/375	2,582
TDEA	70.4%	5,559	6.8%	0.049	0.000	1.084	98.4%	304	-94.5%	8.0%	0.027	0.000	1.153	168/168	6/47	214

* Utils: utilization after physical synthesis; #(Open): number of open sites; TU: track utilization; WNS (ns): worst negative slack; TNS (ns): total negative slack; Power (mW): total power; Δ(Open): reduction of number of open sites; SA: security assets; LSA: locked security assets; LCC: low-controllable cells; LLCC: locked low-controllable cells; KL: key length (bits).

Figure 3: Camellia layout, after initial synthesis (left), locking of security assets (middle), and locking considering timing and controllability (right). The utilization increases from 49.5% to 59.2%, and eventually to 98.9%. Gates introduced by *TroMUX* instances are marked in blue while the other standard cells are filled with grey dots.

MuxLink, we treat each pin of each gate as a node and the connections between pins, including those going through gates, as edges. The connections between the input/output pins of *TroMUX* key-gates are kept as test set for prediction, while all others (except PI or PO connections) are used as training set. We adopt the same graph neural network configuration and training hyperparameters as in [2]; in particular, number of hops for extracting subgraphs is set to 3 and the threshold in post-processing is set to 0.01.

Evaluation Metrics. We report the performance of the *SCOPE* and *MuxLink* attacks using the following established metrics: *accuracy* (AC), *precision* (PC), *key prediction accuracy* (KPA), and *COnstant Propagation Effect* (COPE, [1]); all metrics are in percentage. AC measures the percentage of correctly deciphered key-bits, i.e., $(k_{correct}/k_{total})$. PC measures the correctly deciphered key-bits, optimistically considering every X/undeciphered value as a correct guess, i.e., $((k_{correct} + k_X)/k_{total})$. KPA measures the correctly deciphered key-bits over the entire prediction set, i.e., $(k_{correct}/(k_{total} - k_X))$. COPE measures the vulnerability against the *SCOPE* attack; COPE = 0% means the attack fails entirely.

Results for SCOPE. The results in Table 5 show that 19.35%/19.25% of the key bits are correctly recovered on average. The average value of 0.065% for COPE means that the attack fails almost entirely. Further, the average KPA of 53.72% versus 46.28% for the two keys[5] indicates that *SCOPE* is forced to random guessing for ours. Note that the attack can only decipher a single bit for *TDEA*, thus resulting in KPA of 100% and 0% for that design.

Results for MuxLink. The results in Table 5 show that *MuxLink* can only predict, on average, 20.90% of the key-bits. Moreover, the average KPA is 53.21%, clearly indicating that *MuxLink* is forced to random guessing. In contrast to prior art like *D-MUX* [2, 28]—also showcased in Table 5—ours is superior in thwarting the attack.

6.4 Discussion of Prior Art

Recall the review of related prior art in Sec. 3.1 and the overview in Table 1. Note that none of the prior art released their results

[5] *SCOPE* predicts the two mutually complementary keys, representing the possible configurations of '0' and '1' versus '1' and '0' for key-bits.

Table 5: ML-Based Attack Results on Different Locking Schemes

Design	KL	*SCOPE* [1]						*MuxLink* [2]							
		COPE (%)	*TroMUX* Key 1		*TroMUX* Key 2		#(X)	*TroMUX*				*D-MUX* [2, 28]			
			AC (%)	KPA (%)	AC (%)	KPA (%)		AC (%)	PC (%)	KPA (%)	#(X)	AC (%)	PC (%)	KPA (%)	#(X)
AES_1	1,199	0.1919	28.86	48.39	30.78	51.61	484	28.77	71.48	50.22	512	78.15	79.90	79.54	21
AES_2	1,202	0.1864	31.53	52.71	28.29	47.29	483	28.12	75.04	52.98	564	80.45	81.70	81.47	15
AES_3	441	0.1075	23.36	48.13	25.17	51.87	227	8.16	90.02	45.00	361	88.44	89.12	89.04	3
Camellia	1,271	0.0240	14.16	51.72	13.22	48.28	923	26.12	73.80	49.92	606	90.64	92.60	92.46	25
CAST	1,572	0.0359	18.58	50.52	18.19	49.48	994	37.47	66.03	52.45	449	93.74	94.32	94.29	7
MISTY	1,215	0.1036	23.70	48.90	24.77	51.10	626	33.33	65.84	49.39	395	97.84	98.03	98.02	3
openMSP430_1	1,218	0.0432	19.87	49.90	19.95	50.10	733	23.56	76.60	50.17	646	NA	NA	NA	NA
openMSP430_2	543	0.0306	29.28	52.13	26.89	47.87	238	9.39	90.42	49.51	440	66.85	69.98	69.01	17
PRESENT	241	0.0434	2.49	42.86	3.32	57.14	227	11.62	95.02	70.00	201	NA	NA	NA	NA
SEED	1,612	0.0343	18.55	49.10	19.23	50.90	1,003	37.03	64.02	50.72	435	97.33	97.70	97.70	6
SPARX	2,582	0.0176	21.42	50.23	21.22	49.77	1,481	5.38	94.93	51.48	2,312	NA	NA	NA	NA
TDEA	214	0.0001	0.47	100.00	0.00	0.00	213	1.87	99.07	66.67	208	NA	NA	NA	NA
Avg.	-	0.0682	19.35	53.72	19.25	46.28	-	20.90	80.19	53.21	-	86.68	87.92	87.69	-

* AC: accuracy; PC: precision; KPA: key prediction accuracy; COPE: COnstant Prop. Effect; #(X): number of undeciphered key-bits; NA: Locking using the script in [2] fails.

artifacts publicly. Further, most of the prior art uses different technology libraries and implementation schemes, as well as different benchmarks. Thus, a direct comparison is impractical.

Still, it is essential to recall the considerable limitations of those studies and to forecast related implications. For example, for X(N)OR key-gates utilized by Samimi et al. [22] and Marcelli et al. [19], we note that Alrahis et al. [3] have independently shown that such locking can be circumvented with up to 97.22% accuracy. Thus, we argue that locking as applied by Samimi et al. [22] and Marcelli et al. [19] can be easily circumvented, unlike ours (Sec. 6.3).

7 CONCLUSION

We propose a MUX-based locking scheme, called *TroMUX*, along with a carefully tuned physical-synthesis flow, for preventing targeted Trojan insertion. To the best of our knowledge, ours is the first to systematically employ locking and layout-level means in unison for Trojan prevention. Results on the ISPD'22 contest benchmarks show that ours can reduce the number of open placement sites by 90.3% on average, with security-critical components secured by logic locking. Further, we demonstrate the superior resilience of ours against ML-based attacks: on average, *SCOPE* and *MuxLink* can only correctly predict 19.35%/19.25% and 20.90% of the key-bits, respectively, with around 50% key-prediction accuracy for both, indicating our scheme is enforcing random guessing.

REFERENCES

[1] A. Alaql et al. 2021. SCOPE: Synthesis-Based Constant Propagation Attack on Logic Locking. *Trans. VLSI* 29, 8 (2021), 1529–1542.
[2] L. Alrahis et al. 2022. MuxLink: Circumventing Learning-Resilient MUX-Locking Using Graph Neural Network-based Link Prediction. In *Proc. DATE*. 694–699.
[3] L. Alrahis et al. 2022. OMLA: An Oracle-Less Machine Learning-Based Attack on Logic Locking. *TCS* 69, 3 (2022), 1602–1606.
[4] P.-S. Ba et al. 2015. Hardware Trojan prevention using layout-level design approach. In *Proc. Europ, Conf. Circ. Theory Des.* 1–4.
[5] P.-S. Ba et al. 2016. Hardware Trust through Layout Filling: A Hardware Trojan Prevention Technique. In *Proc. ISVLSI*. 254–259.
[6] P. Chakraborty et al. 2018. SAIL: Machine Learning Guided Structural Analysis Attack on Hardware Obfuscation. In *Proc. AHOST*. 56–61.
[7] X. Chen et al. 2018. Hardware Trojan Detection in Third-Party Digital Intellectual Property Cores by Multilevel Feature Analysis. *TCAD* 37, 7 (2018), 1370–1383.
[8] D. Deng et al. 2020. Novel Design Strategy Toward A2 Trojan Detection Based on Built-In Acceleration Structure. *TCAD* 39, 12 (2020), 4496–4509.
[9] C. Dong et al. 2020. Hardware Trojans in Chips: A Survey for Detection and Prevention. *Sensors* 20, 18 (2020).
[10] S. Dupuis et al. 2014. A novel hardware logic encryption technique for thwarting illegal overproduction and Hardware Trojans. In *Proc. IOLTS*. 49–54.
[11] L. A. Guimarães et al. 2017. Detection of Layout-Level Trojans by Monitoring Substrate with Preexisting Built-in Sensors. In *Proc. ISVLSI*. 290–295.
[12] X. Guo et al. 2019. When Capacitors Attack: Formal Method Driven Design and Detection of Charge-Domain Trojans. In *Proc. DATE*. 1727–1732.
[13] H. Hossein-Talaee and A. Jahanian. 2017. Layout Vulnerability Reduction against Trojan Insertion Using Security-Aware White Space Distribution. In *Proc. ISVLSI*.
[14] Y. Hou et al. 2018. R2D2: Runtime reassurance and detection of A2 Trojan. In *Proc. HOST*. 195–200.
[15] W. Hu et al. 2020. An Overview of Hardware Security and Trust: Threats, Countermeasures and Design Tools. *TCAD* (2020).
[16] R. Karmakar and S. Chattopadhyay. 2020. On Securing Scan Obfuscation Strategies Against ScanSAT Attack. In *Proc. ISQED*. 213–218.
[17] J. Knechtel et al. 2021. Security Closure of Physical Layouts. In *Proc. ICCAD*. 1–9.
[18] J. Knechtel et al. 2022. Benchmarking Security Closure of Physical Layouts: ISPD 2022 Contest. In *Proc. ISPD*. 221–228.
[19] A. Marcelli et al. 2017. An evolutionary approach to hardware encryption and Trojan-horse mitigation. In *Proc. DATE*. 1593–1598.
[20] Nangate Inc 2011. *NanGate FreePDK45 Open Cell Library*. Nangate Inc. http://www.nangate.com/?page_id=2325
[21] P. Ravi et al. 2019. Security is an Architectural Design Constraint. *Microprocess. Microsyst.* 68, C (2019), 17–27.
[22] M. S. Samimi et al. 2016. Hardware enlightening: No where to hide your Hardware Trojans!. In *Proc. IOLTS*. 251–256.
[23] G. Sigl. 2011. Keynote address: Design of secure systems – Where are the EDA tools?. In *Proc. ICCAD*. 1–1.
[24] T. Trippel et al. 2020. ICAS: an Extensible Framework for Estimating the Susceptibility of IC Layouts to Additive Trojans. In *Proc. SP*. 1742–1759.
[25] N. Vashistha et al. 2018. Trojan Scanner: Detecting Hardware Trojans with Rapid SEM Imaging combined with Image Processing and Machine Learning. In *ISTFA*.
[26] A. Vijayan et al. 2020. Runtime Identification of Hardware Trojans by Feature Analysis on Gate-Level Unstructured Data and Anomaly Detection. *TODAES* 25, 4 (2020).
[27] D. Šišejković et al. 2019. Control-Lock: Securing Processor Cores Against Software-Controlled Hardware Trojans. In *Proc. GLSVLSI*. 27–32.
[28] D. Šišejković, F. Merchant, L. M. Reimann, and R. Leupers. 2022. Deceptive Logic Locking for Hardware Integrity Protection Against Machine Learning Attacks. *TCAD* 41, 6 (2022), 1716–1729.
[29] F. Wang et al. 2022. *Baseline and Protected Layouts*. https://drive.google.com/drive/folders/1A_Cy6w2n31_wuPKVayz50R-1lfXfC4vH?usp=sharing
[30] K. Xiao et al. 2014. A Novel Built-In Self-Authentication Technique to Prevent Inserting Hardware Trojans. *TCAD* 33, 12 (2014), 1778–1791.
[31] K. Xiao et al. 2016. Hardware Trojans: Lessons Learned After One Decade of Research. *TODAES* 22, 1 (2016), 6:1–6:23.
[32] K. Yang, M. Hicks, Q. Dong, T. Austin, and D. Sylvester. 2016. A2: Analog Malicious Hardware. In *Proc. SP*. 18–37.
[33] M. Yasin et al. 2017. Security Analysis of Anti-SAT. In *Proc. ASP-DAC*. 342–347.