TroLLoc: Logic Locking and Layout Hardening for IC Security Closure against Hardware Trojans

Fangzhou Wang, Qijing Wang, Lilas Alrahis, Bangqi Fu, Shui Jiang, Xiaopeng Zhang, Ozgur Sinanoglu, Tsung-Yi Ho, Evangeline F.Y. Young, Johann Knechtel

***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 security threats, like piracy of IC intellectual property or insertion of hardware Trojans, i.e., malicious circuit modifications.**

In this work, we proactively and systematically protect the physical layouts of ICs against post-design insertion of Trojans. Toward that end, we propose *TroLLoc*, a novel scheme for IC security closure that employs, for the first time, logic locking and layout hardening in unison. *TroLLoc* is fully integrated into a commercial-grade design flow, and *TroLLoc* is shown to be effective, efficient, and robust. Our work provides in-depth layout and security analysis considering the challenging benchmarks of the ISPD'22/23 contests for security closure. We show that *TroLLoc* successfully renders layouts resilient, with reasonable overheads, against (i) general prospects for Trojan insertion as in the ISPD'22 contest, (ii) actual Trojan insertion as in the ISPD'23 contest, and (iii) potential second-order attacks where adversaries would first (i.e., before Trojan insertion) try to bypass the locking defense, e.g., using advanced machine learning attacks. Finally, we release all our artifacts for independent verification [2].

***Index Terms*—Integrated Circuits; Hardware Security; Trojans; Logic Locking; Physical Design; Security Closure.**

I. Introduction

INTEGRATED circuits (ICs) of varying scale and complexity are at the heart of all our modern-day information systems. An ever-growing body of security threats can compromise information systems in general and ICs in particular [3], [4], [5]. For the design, manufacturing, and deployment of ICs, there are numerous companies and partners involved within complex and world-wide supply chains—ICs run through many hands, where some of those may be acting with malicious intent. Furthermore, once ICs are deployed in the field, adversaries can employ powerful hands-on measures, targeting directly at the hardware and software at runtime.

The resulting cybersecurity challenges are manifold, suggesting the need for holistic and streamlined defense efforts, all the way from software applications down to the IC hardware. Toward the latter end, *electronic design automation (EDA)* algorithms and tools will play an important role [6], [7], [8], [4]. However, state-of-the-art (SOTA) commercial EDA tools do not account yet for such security challenges.

Fangzhou Wang, Qijing Wang, Bangqi Fu, Shui Jiang, Xiaopeng Zhang, Tsung-Yi Ho, and Evangeline F.Y. Young are with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong (e-mail: fzwang@cse.cuhk.edu.hk).

Lilas Alrahis, Ozgur Sinanoglu, Johann Knechtel are with the New York University Abu Dhabi (e-mail: johann@nyu.edu).

A preliminary version of this work has been presented at the ACM International Symposium on Physical Design (ISPD) in 2023 [1].

Security closure is an emerging paradigm that seeks to address this shortcoming of EDA tools. The idea is to proactively harden the layouts of ICs, during physical synthesis, against various threats that are executed post design-time [9], [10], [11]. Such efforts are important for multiple reasons. For example, threats like hardware Trojans or side-channel attacks are explicitly exploiting vulnerabilities of the IC layouts, and any such threats that are not mitigated during design-time will be very difficult to address later on—unlike software, ICs are not easily patchable. See Sec. II-B for more background.

In this work, we focus on the threat of **hardware Trojans**, i.e., malicious circuit modifications that can be introduced by different adversaries throughout the IC supply chain. By design, Trojans are only minor in extent but severe in fallout [12], [13], [14]. Many different types of countermeasures have been proposed over the years. However, most prior art falls short in terms of effectiveness, efficiency, and/or robustness. See Sec. II-A for more background on Trojans and Sec. III for a discussion of relevant prior countermeasures, respectively.

Besides, **logic locking**, or locking for short, means to incorporate so-called key-gate structures that are controlled by secret key-bits [15]. Without knowing the related key-bits required to unlock those key-gate structures, attackers cannot fully understand the structure and functionality of the design at hand. While locking is mainly known for protection of chip-design intellectual property (IP), it can also serve for Trojan defense. See Sec. II-C for more background on locking.

The **objective of this work** is security closure against hardware Trojans. That is, we aim for *proactive, pre-silicon Trojan prevention, by carefully and systematically hardening the physical layout of ICs against post-design Trojan insertion*.

Based on a thorough review of prior art and its limitations, we derive important **research challenges** as follows. Note that various concepts indicated on here, like second-order attacks or open placement sites, are all explained in detail in Sec. II.

- *Robustness – Any Trojan defense must remain robust in place.* That is, a foundry-based adversary, being fully aware that some 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 Trojans. Such second-order attacks represent a key challenge where prior art falls short.
- *Effectiveness – Any defense should protect against various Trojans.* This is especially true for proactive, pre-silicon defenses which have only "one shot" at design-time. More specifically, designs should be protected as a whole, i.e., in terms of (i) layout resources exploitable

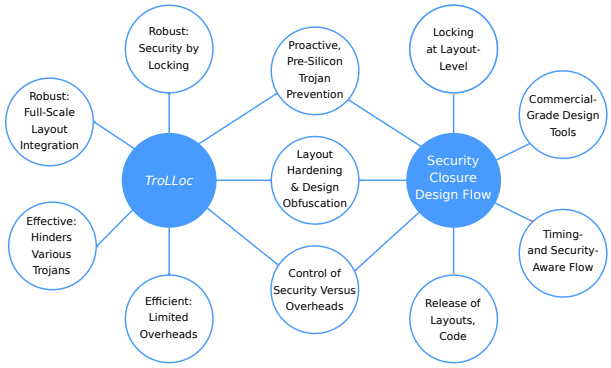

Fig. 1. High-level overview of work. The two main contributions are the locking scheme, *TroLLoc*, and the IC security-closure flow. Both are carefully orchestrated together to achieve pre-silicon Trojan prevention.

for Trojan insertion in general (i.e., open placement sites, free routing tracks, available timing slacks) and (ii) the netlist structure and functionality, which is relevant to hinder Trojans targeting on specific assets. Furthermore, defenses should be evaluated against actual layout-level Trojan insertion, not only against vulnerability estimates.

- *Efficiency – Any defense should incur limited overheads.* Taking control over the underlying trade-offs for resilience against Trojans versus design overheads requires two parts: (i) metrics for security and overheads, and (ii) some integrated, secure-by-design EDA methodology, utilizing those metrics and some user guidance.

To address all these challenges, our work makes the following **contributions**. A high-level representation of our contributions is also given in Fig. 1.

1) Layout-level logic locking – We propose a novel locking scheme, called *TroLLoc*, devised to hinder post-design Trojan insertion. Together with the IC security-closure flow, *TroLLoc* meets important challenges for Trojan defenses: it is robust, effective, and efficient. See further below for an explanation of these achievements.
2) IC security-closure flow – We devise an EDA flow for design-time security closure against post-design Trojan insertion. The flow serves to implement the proposed *TroLLoc* scheme and is fully integrated into a commercial-grade physical-synthesis setup.
3) Layout and security analysis – We conduct a set of thorough case studies that show the promises of the proposed scheme. Toward that end, we are using the real-world benchmark suites from the recent ISPD'22/23 contests on security closure [9], [11]. We are employing a set of SOTA attacks on locking, namely *MuxLink* [16], *OMLA* [17], *Resynthesis* [18], and *SCOPE* [19], and we also conduct actual Trojan insertion as in [11], both to demonstrate the resilience of the proposed scheme against advanced Trojan attacks.
4) Release – We release our secured layouts in [2], to enable independent verification of our work. We will also release our method in full, post peer-review.

Our work on proactive, pre-silicon Trojan prevention is (i) robust, (ii) effective, and (iii) efficient. Regarding (i), we show that advanced adversaries employing (a) second-order attacks using SOTA machine learning (ML)-based attacks on locking or (b) actual Trojan insertion are both hindered. This is a first time in the literature. Regarding (ii), also for the first time, we harden IC layouts against Trojan insertion in general and further protect security-sensitive assets in particular. Regarding (iii), again for the first time, we employ both locking and layout hardening in unison, along with commercial-grade tooling for full control on the security-vs-overhead trade-offs in a real-world IC design and manufacturing setting.

II. Background and Motivation

A. Trojans

1) Real-World Relevance: ICs should provide a root-of-trust platform on which sensitive data operation can rely on. If that premise is broken, however, the confidentiality, integrity, and availability of any application is entirely at risk. For consumer electronics, Trojans that aim to compromise cryptographic circuitry, which exists in modern processors more many years by now, represent such real-world risks [20]. For other domains like military or avionics, the stakes are even higher. Potential Trojan threats on military-grade ICs are discussed in [21], [22]. Remarkable case studies for Trojans in real ICs are described in [23], [14], [24], [20], [25].

2) Working Principles: Trojans are malicious hardware modifications [12], [13]. This concept is quite diverse, covering malicious modifications that may: (i) leak information from an IC, reduce its performance, or disrupt its working altogether; (ii) be always on, triggered internally, or triggered externally; (iii) be introduced through untrustworthy third-party IP, adversarial designers, mask generation, or manufacturing of ICs; etc. Besides, most Trojans require some additional layout-level resources like *open placement sites*, *free routing tracks*, and/or *timing slacks* [9], [11], [26], [27].[1]

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)*[2]—to hinder their detection during testing, rendering them more stealthy—whereas payloads are targeting on sensitive assets like registers holding some key. Note that the trigger and payload components are implemented individually but are working together

[1]Modern IC layouts utilize *standard-cell libraries*, which are provided by foundries and their suppliers, along with the technology-dependent design rules in *technology libraries*. As indicated, the cells in those libraries are standardized, namely in their geometries, such that cells are relatively easy to place into regularly sized and arranged placement rows of an IC. Placement sites are bins of fixed width within such rows. Thus, open placement sites are bins that are not occupied yet by some standard cell. Further, an IC's interconnect comprises a stack of different metal layers; the latter are arranged into regular horizontal and vertical routing tracks. Thus, free routing tracks are stripes within these metal layers that are not utilized yet by some routing shape. Timing slacks can be thought of as performance margins. For an example in simple terms, assume that some part of an IC design, which does exhibit timing slacks, may become slower, e.g., due to insertion of additional Trojan logic. Then, the performance constraints of the overall design can still be met as long as the introduced slow-down is less than the available slacks.

[2]*Controllability* describes whether an end-user can establish specific signal assignments for some internal nets through the IC's primary inputs. Thus, LCNs are nets where end-users are challenged, across a wide range of input patterns, to achieve specific signal assignments and toggling activity.

in tandem. For example, once some LCN is toggling, i.e., only for some rare and carefully crafted condition, the trigger component initiates the actual attack, e.g., the content of some sensitive registers is maliciously forwarded to non-privileged circuit parts or to primary outputs, or even more stealthily leaked through side-channels [20].

3) Threat Modeling: We assume post-design insertion of additive Trojans by adversaries residing within foundries. Among different scenarios (e.g., see [12], [13] for some taxonomies), this one is outstanding as powerful and versatile—related Trojans are not detectable at all during design-time and are also not limited to, e.g., denial-of-service attacks through removal or modification of existing logic, but can embed any malicious functionality at will.

More details on the threat modeling for our work are given in Sec. IV. Besides, an important novel concept for threat modeling in our work is introduced next. While similar ideas may have been used in the literature before, we are the first to explicitly formulated this concept as such.

We differentiate between *first-order* versus *second-order* Trojan attacks. For the former, adversaries would seek to directly insert their Trojans into the IC layout under attack. For the latter, adversaries would first revise the layout to regain layout resources, to aid subsequent Trojan insertion. Both attacks can be supported by means of *engineering change order (ECO)* techniques [20], [28], [29].[3] Besides, in case some layout defenses are put in place, conducting second-order attacks may require adversaries to bypass those as well. Depending on the nature of the defense, it may suffice to again utilize ECO techniques, e.g., to remove *fillers* and/or *spares*,[4] to rearrange the cell placement, etc. More advanced efforts may also be required; e.g., dedicated attacks would be needed to decipher and remove locking structures.

4) Countermeasures and Their Challenges: Trojan countermeasures can be classified into:

1) Proactive, pre-silicon prevention (e.g., [30], [31], [32], [33], [34], [35], [36], [10], [37], [38], [39], [40], [41], [42], [43], [44], [45]);
2) Reactive, post-silicon detection or monitoring (e.g., [46], [47], [48], [49], [50], [51], [52], [53], [45]);
3) Proactive detection, that is (a) pre-silicon inspection/verification (e.g., [13], [54], [55], [56], [57]), (b) post-silicon testing (e.g., [58], [13], [59]), or (c) post-silicon layout inspection (e.g., [60], [61], [25]).

There are various challenges for each of this class as follows. First, many post-silicon schemes rely on a "golden IC," i.e., a Trojan-free reference IC [53]. In threat models where the foundry is untrusted, this requirement is impractical. Second, both post-silicon monitoring and pre-silicon prevention require dedicated hardware features—if not secured properly, the related circuitry itself may be circumvented by adversaries during second-order attacks. Third, by construction, pre-silicon inspection/verification cannot tackle any Trojans inserted by the foundry—such Trojans do not exist yet at design-time. Fourth, post-silicon testing is challenging for multiple reasons: (i) Regular testing procedures may not be helpful to determine some rarely triggered functional or parametric divergences introduced by Trojans. (ii) Trojans can be realized with largely varying logic structures and functional behaviour, making an identification of "common patterns" difficult [62]. While recent ML works like [63], [57] show promise toward that end, they also cannot achieve perfect accuracy. (iii) Process variations as well as measurement noises make the observation of parametric effects introduced by Trojans more and more challenging for advanced IC technologies [64].

Given all these challenges, we argue that a *robust* scheme for proactive prevention is essential to try to hinder Trojans early on and as best as possible. We discuss the relevant prior art in more detail in Sec. III-A and Sec. III-B.

B. Security Closure

As indicated, security closure seeks to proactively harden the physical layouts of ICs at design-time against various threats that are executed post design-time [10], [9], [11]. Security closure against Trojans means to control physical synthesis such that insertion of Trojan components is prevented altogether or at least becomes impractical, while managing the impact on design quality [10], [9], [11], [42], [41], [44]. For example, aggressively dense layouts would leave only few open placement sites and few routing resources exploitable for Trojan insertion [11]. However, while such layouts are already challenging by themselves, in particular for managing *design rule checks (DRCs)*,[5] such arguably naive approach may not be good enough for security closure. This is because, for one, advanced Trojans like *A2* [14] may require only 20 placement sites [26]; that few sites will remain even in any aggressively dense layouts, especially once filler and/or spare cells are considered as exploitable open sites as well. For another, recall that second-order attacks can regain layout resources before Trojan insertion, thereby rendering such implicit protection postulated by dense layouts ineffective in general.

Overall, we argue that efforts for security closure against Trojans need to address the challenges outlined in Sec. I.

C. Logic Locking

1) Working Principles: As indicated, logic locking or locking for short, means to incorporate so-called key-gate structures that are controlled by a secret key [15]. Without

[3]In general, ECO techniques are an industry-wide standard for incremental changes of IC layouts, e.g., to revise the placement and routing of a small module within a larger IC design. Their (mis-)use for Trojan insertion has been successfully demonstrated only recently [20], [28], [11], [29].

[4]Spares are redundant standard cells; they are placed early on but not fully connected yet. Such spares are helpful for last-minute ECO procedures, as their use would likely require only routing-level changes. Fillers are various non-functional cells that serve different purposes, e.g., acting as decoupling capacitors (decaps) to stabilize the IC's internal power supply network. Although spares and fillers can be used for pre-silicon prevention, by simply filling up all open placement sites with them, it is important to note that those components are easy to exploit for Trojan insertion as most of them can be readily removed without disrupting the core functionality of an IC [11], [25].

[5]DRCs describe technology rules for manufacturability of ICs, in terms of cell placement (e.g., considering the so-called *N-well continuity* which is essential for IC semiconductors) and in terms of routing shapes (considering minimum areas, pitches, et cetera). Thus, any violation for DRCs can disqualify the whole IC from manufacturing. For modern ICs and their advanced technologies underlying, DRCs are becoming more and more complex [65].

TABLE I
HIGH-LEVEL COMPARISON WITH PRIOR ART FOR PRE-SILICON TROJAN PREVENTION

Works	Approach	Robust	Effective Protection of: Functionality / Layout	Efficient	Tested against Actual Trojans	Artifacts Available
[30]	Locking	✗ [30], [32]	✗^{a} / ✗	✓✗	✗	✗
[32], [31]	Locking	✗✓ [17]	✗✓^{a} / ✗	✓✗	✗	✗
[33]	Locking	?	?^{a} / ✗	✓✗	✗	✗
[34], [35], [36], [45]	Additional Logic	✓✗ [20], [28], [11]	✗ / ✓✗^{a}	✓✗	✗	✗
[37], [10], [41], [42], [43], [44]	Layout Hardening	✗✓ [20], [28], [11]	✗ / ✗✓^{a}	✓✗	✗	✓✗
[40]	Routing	✓	✗ / ✓✗^{b}	✓	✓✗ [14]	✗
Ours: *TroLLoc*	**Locking and Layout Hardening**	✓	✓ / ✓	✓	✓ **[11]**	✓ **[2]**

Symbols: ✓ – yes, ✗ – no, ? – unclear, ✓✗ – yes but with some caveats, ✗✓ – unlikely. Notes: a – Lack or uncertainty of robustness is expected to undermine the real-world effectiveness. b – Covers only routing resources.

the full knowledge of the key, logic locking ensures that the details of the design IP cannot be fully recovered and the IC remains non-functional. The key-gate structures are commonly realized using, e.g., X(N)OR gates [66], AND/OR gates [30], look-up tables (LUTs) [67], or multiplexers (MUXes) [68]. After manufacturing, preferably only after testing [69], the IC is activated: the secret key is loaded into a dedicated, on-chip tamper-proof memory by a trusted party.

While locking is largely known for protection of design IP [15], it can also serve for Trojan defense. In fact, different such schemes have been proposed: AND/OR locking [30], X(N)OR locking [32], [31], and custom locking [33]. We discuss relevant prior art in more detail in Sec. III-A.

2) Threat Modeling: Over the years, various attacks against the different implementations of locking have been proposed. Two different threat models are considered:[6]

- The *oracle-guided* model assumes access to an activated IC (i.e., with the correct key loaded), which can be purchased from the market, yet only after manufacturing. Using another IC copy, the attacker also has to extract the locked gate-level netlist through means of reverse engineering. The attacker then aims to decipher the key by analyzing the netlist, identifying meaningful input patterns through analytical means such as Boolean satisfiability (SAT) solving [70], applying those to the working IC, and observing the responses to iteratively converge toward the correct key.
- The *oracle-less* model assumes only access to the locked gate-level netlist. The attacker aims to decipher the key by analyzing the structure and functionality of the netlist, e.g., by means of ML [16], identifying meaningful patterns and correlations of key-gates and correct key-bits.

3) Advanced Oracle-Less Attacks: Recently, ML-based attacks like *SAIL* [71], *SCOPE* [19], *OMLA* [17] and *MuxLink* [16], as well as structural attacks like *Resynthesis* [18], succeeded in identifying various relevant design features in locked circuits and in subsequently predicting key-gate structures, all in an oracle-less setting.

[6]For our work, we only have to consider the oracle-less model. This is because we seek to protect against post-design Trojan insertion by the foundry, where oracle ICs are not available yet. More details on the threat modeling for our work are given in Sec. IV.

For example, using a graph representation of the IC design, *MuxLink* [16] learns on the structure of regular, unlocked parts of a design at hand that is specifically locked using MUX key-gates. After training, *MuxLink* seeks to decipher the input connections to 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 bypassing the key-gate structure—and contrasts the resulting different outcomes to the trained knowledge to predict the more likely original structure, i.e., the circuitry before locking.

For another example, *Resynthesis* [18] systematically resynthesizes the locked netlist across a large range of varying synthesis parameters like mapping efforts and timing constraints, the latter especially for paths related to key-gate structures. All resulting netlists, which are structurally different yet functionally equivalent, are subsequently passed to other attacks like *SCOPE*, upon which majority voting is conducted across all the resulting inferences for each key-gate structure.

Given their high success rates, such oracle-less attacks represent a considerable threat for locking. Especially for locking-based, proactive Trojan prevention at design-time, such attack capabilities need to be taken in consideration. Otherwise, second-order attacks would remain a practical threat.

III. PRIOR ART AND THEIR LIMITATIONS

Next, we discuss the prior art for proactive, pre-silicon Trojan prevention in more detail, including their limitations. We differentiate into locking-based versus layout-based schemes.

A high-level comparison of prior art and ours is also outlined in Table I. It is important to note that our work addresses *all* the limitations identified for prior art.

A. Locking-Based Trojan Prevention

Dupuis et al. [30] 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. [32] follow similar principles as Dupuis et al., but utilize X(N)OR key-gates. Marcelli et al. [31] propose a multi-objective evolutionary algorithm, seeking to minimize LCNs and maximize the efficacy of X(N)OR locking at the same time. Šišejković et al. [33] secure inter-module

control signals against software-controlled hardware Trojans, using encryption circuitry along with regular locking. However, they do not specify/limit the type of key-gates.

The above prior art has limitations as follows. First, Dupuis et al. [30] cannot protect specific assets against Trojan insertion. This is because they utilize AND/OR key-gates, to tune the controllability of selected nets, but not to obfuscate the design; see also the next point. Second, none show robustness against SOTA oracle-less attacks on locking. For example, Dupuis et al. [30] do not employ resynthesis after locking which, however, would be essential to obfuscate the correlation between key-gate structures and key-bit assignments [15]. Third, Šišejković et al. [33] can only protect against Trojans targeting on inter-module control signals, not on any of the modules' internal circuitry. Accordingly, Trojan attacks seeking to, e.g., manipulate certain computation steps, can still succeed. Fourth, none consider Trojans at the layout level; all are working at netlist level. This implies that none can conclusively prevent post-design insertion of Trojans in the real world. Finally, none explicitly prevent Trojan payloads. Šišejković et al. [33] are locking the design in general, without focus on triggers or payloads, and others are locking only LCNs to hinder the integration of Trojan triggers.

B. Layout-Based Trojan Prevention

Xiao et al. [34] propose to add built-in self-test components, arguing that tampering of those structures (e.g., by adversaries trying to regain layout resources for their Trojans) can be detected during post-silicon testing. Similarly, Ba et al. [35], [36] and Eslami et al. [45] insert other additional circuitry. Hossein-Talaee et al. [37] redistribute whitespace / open placement sites otherwise exploitable for Trojan insertion. Trippel et al. [40] employ specific routing structures, to prevent routing of Trojan components. Knechtel et al. [10], Guo et al. [41], Hsu et al. [42], Wei et al. [44], and Eslami et al. [43] all seek to harden layouts through physical synthesis. For example, Knechtel et al. propose to locally increase placement density (to hinder insertion of Trojans) and to locally increase routing density (to hinder routing of Trojans), while Eslami et al. propose a holistic synthesis methodology; they demonstrate their work with an actual IC tape-out as well.

The above prior art has limitations as follows. First, none can truly protect specific assets against Trojans. In the absence of locking or other obfuscation schemes, the original design is fully accessible by adversaries. Second, except for Trippel et al. [40], none show robustness against second-order attacks. For example, for Hossein-Talaee et al. [37], shifting of whitespaces could be trivially reverted by ECO-assisted Trojan insertion [20], [28], [29]. While the works of [34], [35], [36] aim toward such robustness, the use of additional circuitry by itself does not resolve the fact that other parts of the IC design may remain exploitable. In particular, fillers and spares are utilized in most if not all prior art and, as indicated in Sec. II-A3, such components are easy to exploit. Third, for full efficacy, the work by Xiao et al. [34] requires 100% test coverage which can be very difficult to achieve for real-world, large-scale designs. Further, high utilization rates are challenging for their method. Both aspects are limiting the practicability of the scheme. Fourth, for the works in [34], [35], [36], the number of additionally required primary inputs scales with layout hardening. 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. Finally, except for Trippel et al. [40], none have evaluated the resilience of their proposed schemes against insertion of actual Trojans. Note that Trippel et al. consider only one specific Trojan, namely *A2* [14]; their evaluation cannot provide insights for any other kinds of Trojans.

IV. Threat Modeling

1) Trojans: We base our assumptions on classical threat modeling for post-design insertion of additive Trojans [12], [23], [26], [40], [20]. Details are given next.

1) We assume adversaries reside within foundries, whereas the design process is trustworthy.
2) From 1) follows that adversaries have no knowledge of the original design, only of the physical layout at hand, which is protected through *TroLLoc*. Adversaries also have full knowledge of the IC technology.
3) In an advancement over prior art, we do not utilize any spares or fillers in our protected layouts. Thus, naive Trojan attacks exploiting such resources are ruled out.
4) From 1) also follows that we do not account for Trojans introduced by, e.g., malicious third-party IP modules. Orthogonal countermeasures, like reactive monitoring (Sec. II-A4), may be applied against such threats as well, in conjunction to ours.
5) Trojans are assumed to be implemented using regular standard cells, not by modifying transistors, etc.
6) We assume Trojans with triggers based on LCNs and payloads targeting on specific *assets*, i.e., security-critical components like registers holding some key.
7) The adversaries' objective is to insert some Trojan(s), more specifically some dedicated trigger and payload components, into the physical layout. Toward that end, adversaries may follow first-order attacks, i.e., direct insertion of Trojan components in the layout as is, or second-order attacks, i.e., revise the layout as much as necessary to insert the Trojan, as long as the layout then still meets DRCs and other final design checks.[7]
8) Related to 7), recall that attackers cannot exploit fillers and/or spares, as our *TroLLoc*-protected layouts do not contain such components.

2) Locking: Recall that we employ locking exclusively to hinder Trojan insertion. Thus, we apply oracle-less threat modeling [16], [71], [19], [17], [18], in consistency with the above threat modeling for Trojans, as follows.

1) Adversaries are assumed to reside within foundries. Thus, only oracle-less attacks are applicable.

[7]While reverse engineering and/or reactive monitoring may help to detect the related layout modifications introduced by second-order attacks—as well as to detect Trojans in general—we do not consider such orthogonal post-silicon techniques as part of our work.

2) From 1) follows that adversaries have no knowledge of the original design, only of the physical layout at hand, which is protected through *TroLLoc*. Adversaries also have full knowledge of the IC technology.
3) Following Kerckhoffs' principle, all implementation details for *TroLLoc* are known to the adversaries; only the secret key-bits remain undisclosed.
4) The adversaries' objective is to circumvent the locking scheme, to (a) understand the true functionality of the design, which is required for attacking specific assets, and (b) to regain layout resources, which is required for Trojan insertion in general.

V. Methodology

Recall that we seek to devise a scheme for proactive, pre-silicon Trojan prevention as an addition/alternative to only reactive, post-silicon Trojan detection. Also recall that related prior art all falls short in terms of robustness, effectiveness, and/or efficiency. Our approach—which can be summarized as locking and layout hardening applied carefully in unison—aims to hinder both first-/second-order attacks of post-design Trojan insertion. For locking, we devise a scheme which is (i) resilient against various SOTA attacks and (ii) directly integrated at layout-level. For layout hardening, unlike prior art, we neither employ trivial filler/spare cells nor any other vulnerable circuitry but only locking instances. To protect IC layouts holistically, our methodology carefully embeds, in a security-aware manner, as many locking instances during physical synthesis as practically possible, i.e., while keeping the design quality well under control.

Next, we propose the locking scheme, *TroLLoc*. Thereafter, we devise its integration into a security-and-design-aware synthesis flow based on commercial-grade IC tooling.

A. TroLLoc

Locking Approach: First, note that both MUX and X(N)OR gates are reasonable choices for key-gates in locking schemes. In fact, both types are used extensively in prior art, with specific attacks being tailored for each type (Sec. II-C). Thus, we consider both types for implementation as well as all the experimental investigation. For brevity, we will refer to "*TroLLoc* implemented with MUXes as key-gates" as *TroLLoc-MUX* and "*TroLLoc* implemented with X(N)ORs as key-gates" as *TroLLoc-XOR* in the remainder of this paper.

Unlike prior art for locking-based Trojan prevention (Sec. III-A), *TroLLoc* utilizes key-gates in a similar way as done in SOTA locking schemes, with the specific intent to avoid information leakage arising from the key-gates' structure and connectivity. *TroLLoc* employs simple but resilient key-gate structures with localized connectivity, described next.

Learning-Resilient Key-Gate Structures: *TroLLoc* ensures that key-bits can be fully randomized, without inducing correlations to the locked gate's true functionality. Thus, *TroLLoc* shall be resilient against ML-based attacks by construction. To do so, *TroLLoc* instances render any locked gate interchangeable with their complementary counterparts, e.g., a locked NAND gate can act as AND or NAND, only depending on the key-bit. More specifically, one picks randomly from the different possible configurations applicable for *TroLLoc-MUX* or *TroLLoc-XOR*, as shown in Fig. 2.

Continuing the example of AND/NAND gates, for *TroLLoc-MUX*, one would consider the four configurations in the upper part of Fig. 2(a). Note how these configurations are pairwise indistinguishable. Also note how the two configurations to the right are based on transforming the original gates to their counterparts—this serves for further obfuscation of the overall layout regarding the distribution of gate types. For *TroLLoc-XOR*, aside the random transformation of original gates, we also randomly select XOR/XNOR key-gates.

Concerning flip-flops (FFs), these can be locked as is, by connecting the Q and/or QN ports[8] to a randomly selected, appropriate key-gate structure shown in Fig. 2(b, c).

No Further Information Leakage from Physical Layouts: Aside from the fact that *TroLLoc* key-gate structures are pairwise complementary, which similarly applies to most if not all locking schemes, there are *no* direct correlations, neither between the types of original versus locked gates, nor between the locked gates and the correct key-bits.

Here it is also important to note that, for *TroLLoc-MUX*, all internal nets connecting to/from the MUX key-gate share a common timing path and are, thus, jointly optimized by the synthesis engine. Accordingly, an attacker seeking to extract any additional information from the underlying timing paths, driver strengths, etc., cannot succeed. For *TroLLoc-XOR*, this fact applies similarly as well.

B. Security-and-Design-Aware Physical Synthesis

Next, we introduce our physical-synthesis flow that is capable of hardening any post-route layout with *TroLLoc* instances, incurring well-controlled timing and area overheads in the process. As outlined in Fig. 3, we start with the original netlist and other necessary data to generate a baseline layout and then carefully interleave locking and physical synthesis throughout two stages to produce a final protected layout.

1) Two-Stage Locking Scheme: In the first stage, we initially lock all security-critical FFs, i.e., FFs defined as assets by the user, with *TroLLoc* instances. After locking, we conduct placement and routing (P&R), and obtain a partially protected layout (PPL). Note that a PPL has only assets protected; no other parts like LCNs are protected yet. Further, depending on the number of user-provided assets, a PPL can still exhibit a relatively low utilization, potentially leaving the layout vulnerable to Trojan insertion.

Thus, in the second stage, we aim at locking as many more components as practically possible, while prioritizing the locking of LCNs, all without degrading the design quality. Toward that end, we start by extracting some specifics obtained from the PPL: remaining open sites, timing paths, and controllability of all nets. Then, we derive the number of cells to be locked considering open sites and timing; this step is detailed in

[8]For circuit-level efficiency, standard-cell libraries typically contain two types of FFs: those with the stored data provided as complementary pair of output signals, called Q and QN, and those with only one output signal, either for the original data Q or the inverted data QN. Design tools then automatically instantiate, for each FF, the type which is most suitable/efficient.

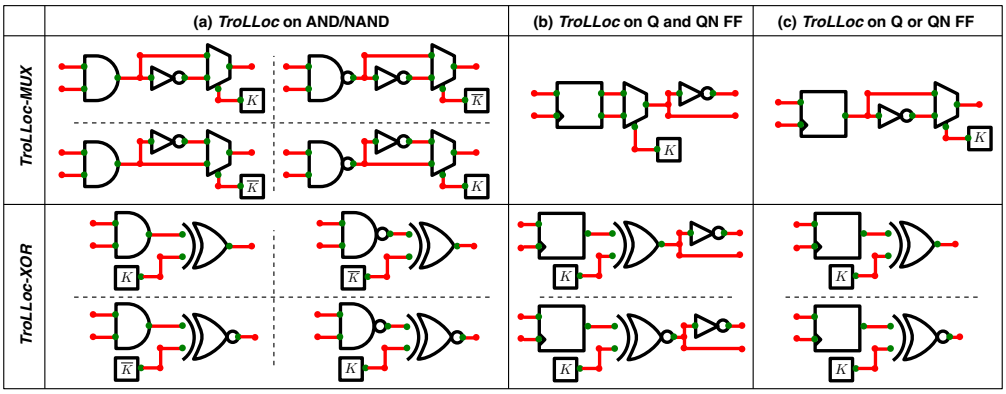

Fig. 2. Design of *TroLLoc* instances. For *TroLLoc-MUX* or *TroLLoc-XOR*, respectively, the key-bit is connected to the MUX select line or an XOR/XNOR input. (a) Locking of simple AND/NAND gates. Note that other simple gates are locked similarly, which is not illustrated here. When $K = 0$, all *TroLLoc* instances operate as AND gate; when $K = 1$, they function as NAND gate. Therefore, given any such *TroLLoc* instance, its functionality will remain unknown without the correct key-bit. (b, c) Locking of different types of FFs. For (b), the key-bit dictates which output signal holds Q and which QN; for (c), the key-bit dictates whether Q or QN is put out. For *TroLLoc-XOR*, we randomly pick only one of the FF's outputs and randomly pick XOR or XNOR as key-gates.

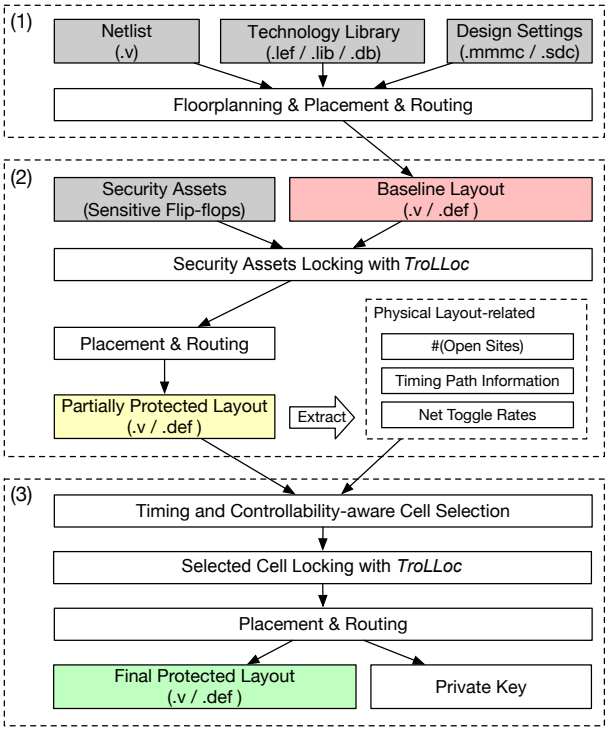

Fig. 3. Our physical-synthesis flow. (1) Initial synthesis of baseline layouts. (2) Locking of security assets, followed by synthesis. (3) Locking of further components, considering timing and controllability, followed by synthesis.

Sec. V-B3. Next, we perform cell selection considering timing and controllability; this step is detailed in Sec. V-B4. Then, all selected cells are locked. After that, P&R is conducted again, and we retrieve a highly-utilized, final protected layout.

Finally note that initial experiments had shown that this two-stage approach is more feasible and efficient, as in more timing-friendly, over a single-stage approach. That is, when we did directly use the specifics extracted from the baseline layouts (marked in red in Fig. 3) for cell selection and locking, it was much harder to achieve timing closure, due to an accumulation of slack estimation errors. As indicated, more details on the timing-and-controllability-driven cell selection in the second stage are provided in Sec. V-B4.

2) Storage of Key-Bits: For each *TroLLoc* instance, there will be one corresponding key-bit, requiring some facility to store all the key-bits. We employ a large shift register for that, without loss of generality but for the following reasons:

- Prior art often considers adding a dedicated PI for each key-bit, which is not scalable. For our, only two additional PIs are required (data in, load) for any key length.
- Such shift register incurs negligible impact on performance and power. For an IC locked with k key-bits using *TroLLoc*, loading the key will take k clock cycles. This is done once during the initial boot-up, while the main circuitry is still hold in reset. Once the load signal is reset to `low`, the key-bits will remain stable and all related FFs are only consuming some static power.
- The related FFs are also helpful for locally filling/hardening open placement sites, whereas bulky memory blocks would make this task rather complicated.

3) On-Demand Key Length: *TroLLoc* instances help to occupy open placement sites with their INVs, MUXes/X(N)ORs, and FFs. For *TroLLoc-MUX* for example, we determine the key length for the second stage 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. For *TroLLoc-XOR*, we determine the key length similarly.

For α, note that commercial IC tools conduct timing optimization by gate sizing, repeater insertion, etc. Thus, we need to reserve some space for such optimization efforts during locking. Accordingly, for designs where timing closure is more challenging, we will utilize a larger α value, and vice versa.

4) Timing and Controllability-Aware Cell Selection: Since locking more and more components through *TroLLoc* instances will introduce further delays to all related timing paths, the selection of cells to lock becomes critical for timing clo-

TABLE II
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 average nr. of toggles per clock cycle for n

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: $C' \leftarrow \{\}$;
2: **while** $|C'| < K$ **do**
3: **foreach** cell $c \in C$ **do**
4: $c.score \leftarrow cellScore(c, N, P)$; ▷ 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 $n \in N(c_h)$ **do**
9: **foreach** timing path $p \in P(n)$ **do**
10: $p.slack \leftarrow p.slack - \sigma$; ▷ Pessimistic slack estimation
11: **return** C';

sure. Thus, we propose a scoring function $cellScore(c, N, P)$ to comprehensively describe the priority of a cell c as follows:

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

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

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

with terms described in Table II. More details are given next.

First, the score is represented as sum of net-level scores to generalize to multi-output cases, e.g., both Q and QN of a FF are used. Second, the net-level score is devised to jointly consider timing and controllability. By using a sigmoid function, the net-level score value remains positive even for negative but small slacks. Third, 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. Fourth, when calculating MS, some nets may not be covered by any of the reported timing paths—this scenario in general is a well-known limitation of commercial IC tools. For those nets, $MS(n, P)$ returns -0.5 as a "fall-back value" which is close to observed average of negative slacks. Thus, this value serves to conservatively penalize cells with unknown timing.

Using the above scoring, we propose an iterative cell-selection procedure in Algorithm 1. There, in each iteration, we calculate the cell scores (line 3–4) and pick the cell(s) with the highest score (line 5–7). Next, we update the recorded slacks of all affected timing paths based on σ, i.e., a pessimistic estimate of delays introduced by *TroLLoc* instances. For *TroLLoc-MUX* (or *TroLLoc-XOR*), σ is defined as the sum of worst-case delays for `INV_X1` and `MUX2_X1` (or `XOR2_X1`), which are the default options for actual cells for *TroLLoc* instances. Note that the worst-case delays are derived for the matching corners (line 8–10).[9]

VI. EXPERIMENTAL INVESTIGATION

A series of thorough case studies is conducted that demonstrates the robustness, effectiveness, and efficiency of the proposed scheme. More specifically, in Sec. VI-B, we provide a layout and security analysis, showing that *TroLLoc* can serve to harden layouts against Trojan insertion with reasonable overheads. In Sec. VI-C, actual layout-level insertion of various Trojans is conducted, showing that *TroLLoc* is effective and robust against such real-world attacks, which were largely overlooked in prior art. Furthermore, in Sec. VI-D and Sec. VI-E, respectively, we demonstrate the resilience of *TroLLoc-MUX* and *TroLLoc-XOR* against second-order attacks where advanced adversaries would first try to bypass the locking defenses before conducting Trojan insertion.

A. General Setup

Tools: As indicated, we base our work on a commercial-grade design flow. For experiments on *TroLLoc-MUX* and *TroLLoc-XOR*, respectively, *Innovus 20.14* and *Innovus 21.14* (by *Cadence*) are used for physical synthesis.[10] The methodology is implemented in custom *TCL* scripts and *Python* code.

Benchmarks and Technology Library: We employ the ISPD'22 contest benchmark suite on security closure [9]. This suite contains the baseline *DEF* layout files, the corresponding post-route *Verilog* netlists, the *SDC* and *MMMC* files used for timing analysis, the customized *LIB/LEF* files for the well-known and publicly available *Nangate 45nm Open Cell Library* [72], and custom files describing the security assets. Since the suite was originally synthesized by some legacy versions, namely *Innovus 18.13* and *Innovus 16.15*, we initially resynthesize all designs at our end. We do so for floorplan utilization rates similar to the original baseline layouts; only the rates for *AES_3*, *openMSP430_2*, and *TDEA* are set 10% lower, as needed to lock all security assets.

As Table III shows, the designs vary in terms of complexity, utilization, size (for both cells and nets), available metal layers, timing constraints, and considered corners.

B. Case Study 1: ISPD'22 Benchmarks

We quantify security and layout results for the resynthesized baseline layouts versus the final, protected layouts in Table IV. In Fig. 4, we provide an example for *TroLLoc-MUX* applied on the *Camellia* benchmark. More details are discussed next.

[9] *Corners* describe different sets of parameters for characterizing an IC's operation and performance under varying ambient conditions.

[10] The use of different versions is merely due to the fact that these two different sets of experiments were conducted at different points in time. We confirmed that only negligible differences in design quality arise, e.g., differences in utilization are within 1%; thus, these two versions of *TroLLoc* remain compatible and results as well as findings are comparable.

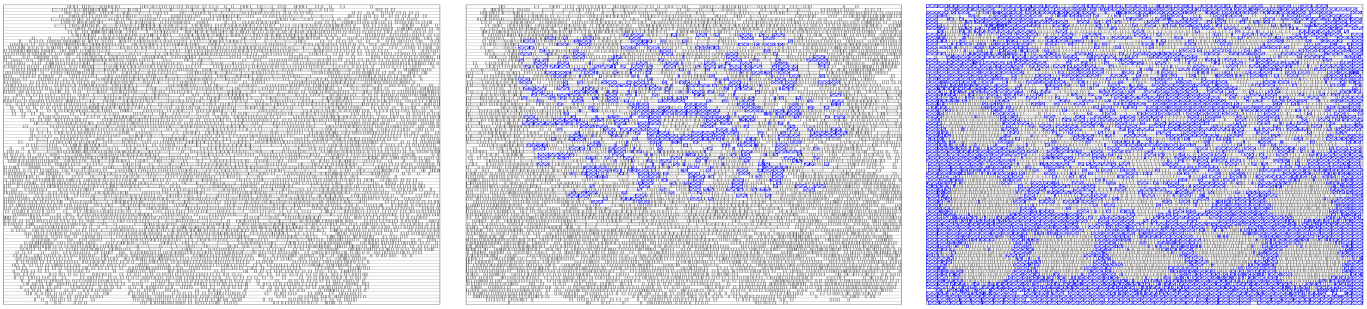

Fig. 4. Layout illustrations for the benchmark *Camellia* under *TroLLoc-MUX*: (left) baseline layout, i.e., after initial resynthesis; (middle) after first-stage locking of security assets, (right) final protected layout, i.e., after second-stage locking of other cells considering timing and controllability. Cells introduced by *TroLLoc-MUX* instances are marked in blue, whereas others are marked in grey. The utilization increases from 49.5% (left) to 59.2% (middle), and eventually to 98.9% (right); also recall Footnote 11. Note that layouts heights are reduced here; the original aspect ratio is 1.0.

TABLE III
STATISTICS FOR ISPD'22 BENCHMARKS

Design	F. Utils	#(Cells)	#(Nets)	#SA	#(ML)	CP	Corner
AES_1	75.0%	16,509	19,694	291	10	1	typical
AES_2	75.0%	16,509	19,694	291	10	1	typical
AES_3	85.0%	15,836	19,020	291	10	1	typical
Camellia	50.0%	6,710	7,160	256	6	10	slow
CAST	50.0%	12,682	13,057	192	6	10	slow
MISTY	50.0%	9,517	9,904	204	6	10	slow
openMSP430_1	50.0%	4,690	5,312	340	6	30	slow
openMSP430_2	70.0%	5,921	6,550	334	6	8	slow
PRESENT	50.0%	868	1,046	80	6	10	slow
SEED	50.0%	12,682	13,057	195	6	10	slow
SPARX	50.0%	8,146	10,884	2,176	6	10	slow
TDEA	70.0%	2,269	2,594	168	6	4	slow

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

First, all the protected layouts exhibit much higher utilization rates: 93.6–99.0%.[11] Recall that, unlike prior art, we do not utilize any fillers or spares toward that end. This achievement helps to significantly reduce open placement sites, namely by 90.3% and 89.5% on average, respectively, for *TroLLoc-MUX* and *TroLLoc-XOR*. In turn, this renders all layouts more resilient against Trojan insertion—the less open sites remain, the more difficult Trojan insertion becomes [9], [26]. We also achieve higher routing track utilization (defined as *total routed wire length* / *total track length*); thus, routing for Trojans will become more challenging as well [9], [26].

Second, it is essential to note that, despite the much higher utilization rates achieved without using any filler or spares, all protected layouts are without DRC violations. This proves the efficiency and real-world relevance of our proposed scheme, in particular of the EDA flow for security closure. In fact, we can tune the flow to achieve this very trade-off: largest possible utilization without introducing DRC violations. Doing so is a well-known challenge for IC design in general [65], and our method is fully supporting the security-concerned IC designer with automated guidance and feedback.

When considering the two findings above together, the following important **research question** arises: *facing such layout hardening, can adversaries still succeed in exploiting the remaining open sites for Trojan insertion, or will they be hindered by DRC violations and/or other real-world design challenges?* To answer this, as indicated, we conduct actual Trojan insertion on the hardened layouts in Sec. VI-C.

Finally, for overheads induced by *TroLLoc*, we note the following. Total power is increased by 18.5% for *TroLLoc-MUX* and by 17.7% for *TroLLoc-XOR* on average, respectively. This is reasonable given all the additional cells introduced through *TroLLoc* instances. Concerning performance, the average and the maximal worst negative slacks (WNS)[12] are 1.41% and 3.10% for *TroLLoc-MUX*, and 0.54% and 1.40% for *TroLLoc-XOR*, respectively. This is reasonable as well, again, given all the additional *TroLLoc* circuitry. It is also important to note that, unlike DRCs or other critical design checks, such WNS violations do *not* render the IC unfit, neither for manufacturing nor for actual use. The only requirement for such ICs would be to operate them with an accordingly reduced clock period.

C. Case Study 2: ISPD'23 Trojan Insertion

Setup: Here we also employ parts of the ISPD'23 suite [11], namely the actual Trojans. Note that we refrain from using the ISPD'23 suite in full here—for a fair evaluation of the attackers prospects, i.e., to put the related outcomes for Trojan insertion correctly into the context of the prior security and layout analysis in Sec. VI-B, we have to continue working with the ISPD'22 benchmarks here.

Only a subset of Trojans are readily compatible with the ISPD'22 suite, namely those targeting on the very same cell instances for the respective benchmarks. We obtain all 9 related Trojans by performing technology mapping at our end. Also note that we employ the very same ECO techniques for Trojan insertion as in the ISPD'23 contest; we thank the ISPD'23 organizers for sharing the related scripts with us.

[11] In general, the achievable utilization is subject to the IC design and the technology library. For our work, showcasing such ultra-high utilization is important—it supports our claim of truly hardening layouts against Trojan insertion. As indicted, prior art like [34] is challenged by high utilization.

[12] WNS quantifies the worst timing path, i.e., the path with the largest negative value and, thus, the path that exhibits the largest timing violation. Any WNS number is to be interpreted in context of the clock period.

TABLE IV
LAYOUT AND SECURITY RESULTS 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*	*KL*
TroLLoc-MUX	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	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	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	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	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	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	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	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	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	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	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	2,582
	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	214
TroLLoc-XOR	AES_1	74.9%	44,781	8.5%	0.000	-0.001	58.671	96.1%	6,922	-84.5%	10.7%	-0.014	-0.160	59.689	1,200
	AES_2	74.7%	45,152	8.5%	0.000	0.000	58.766	95.0%	8,971	-80.1%	10.6%	-0.007	-0.057	59.554	1,149
	AES_3	86.3%	21,244	9.8%	-0.004	-0.011	59.727	95.5%	6,921	-67.4%	10.7%	-0.003	-0.042	60.719	416
	Camellia	49.5%	33,930	9.2%	0.856	0.000	1.237	98.7%	864	-97.5%	12.2%	0.075	0.000	1.478	1,348
	CAST	49.1%	54,501	8.9%	0.023	0.000	3.143	94.6%	5,774	-89.4%	11.7%	-0.094	-0.934	3.867	1,804
	MISTY	48.5%	43,365	8.7%	-0.013	-0.013	2.266	93.9%	5,102	-88.2%	11.8%	-0.097	-1.554	2.945	1,315
	openMSP430_1	49.6%	32,813	6.1%	0.000	0.000	0.375	98.7%	875	-97.3%	10.1%	0.000	0.000	0.542	1,337
	openMSP430_2	70.4%	15,170	7.7%	-0.004	-0.008	1.238	97.4%	1,349	-91.1%	10.0%	0.000	0.000	1.371	582
	PRESENT	49.8%	6,274	4.2%	5.654	0.000	0.197	97.4%	327	-94.8%	6.2%	4.994	0.000	0.234	255
	SEED	49.1%	54,501	8.9%	0.023	0.000	3.143	94.1%	6,364	-88.3%	11.6%	-0.024	-0.106	3.745	1,802
	SPARX	49.8%	70,038	6.2%	2.389	0.000	2.161	99.0%	1,419	-98.0%	13.3%	0.319	0.000	2.888	2,746
	TDEA	70.0%	5,634	6.8%	0.021	0.000	1.083	99.0%	185	-96.7%	8.1%	0.017	0.000	1.173	226

Utils: utilization after physical synthesis; #(Open): nr. of open sites; TU: track utilization; WNS (ns): worst negative slack; TNS (ns): total negative slack; Power (mW): total power; Δ(Open): reduction of nr. of open sites; KL: key length (bits), accounting both for locked assets and other locked cell instances.

TABLE V
RESULTS FOR ISPD'23 TROJAN INSERTION ON THE ISPD'22 CONTEST BENCHMARK SUITE

	Design – Trojan / {Trojans}	Baseline Layout (Resynthesized)							Protected Layout (Final)						
		Mode 'reg'		Mode 'adv'		Mode 'adv2'		*Res*	Mode 'reg'		Mode 'adv'		Mode 'adv2'		*Res*
		#(Vios)	*TNS*	*#(Vios)*	*TNS*	*#(Vios)*	*TNS*		*#(Vios)*	*TNS*	*#(Vios)*	*TNS*	*#(Vios)*	*TNS*	
TroLLoc-MUX	Camellia – {burn random, burn targeted}	0	0.000	0	0.000	0	0.000	✗	11	0.000	21	0.000	89	-3.022	✓✗
	Camellia – fault random	28	0.000	0	0.000	0	0.000	✗	95	0.000	33	0.000	26	-0.189	✓✗
	Camellia – fault targeted	19	0.000	0	0.000	0	0.000	✗	54	-0.102	28	0.000	51	-2.611	✓✗
	Camellia – {leak random, leak targeted}	0	0.000	0	0.000	0	0.000	✗	62	0.000	61	0.000	2,737	-99.210	✓✗
	SEED – burn random	0	-3.265	0	-3.293	0	-3.184	✓✗	0	-2.249	0	-2.224	40	-2.395	✓✗ ⊖
	SEED – fault random	12	-3.633	0	-3.624	0	-3.353	✓✗	35	-2.168	18	-2.326	24	-2.045	✓✗ ⊕
	SEED – leak random	0	-3.177	0	-3.188	0	-3.290	✓✗	0	-2.137	0	-2.199	11	-4.370	✓✗ ⊖
TroLLoc-XOR	Camellia – {burn random, burn targeted}	0	0.000	0	0.000	0	0.000	✗					146	-0.222	✓✗
	Camellia – fault random	40	0.000	0	0.000	0	0.000	✗					164	-0.769	✓✗
	Camellia – fault targeted	20	0.000	0	0.000	0	0.000	✗					95	-0.002	✓✗
	Camellia – {leak random, leak targeted}	0	0.000	0	0.000	0	0.000	✗					300	-2.531	✓✗
	SEED – burn random	0	-1.188	0	-1.312	0	-1.170	✓✗	0	-0.750	3	-0.798	10	-7.123	✓✗ ⊖
	SEED – fault random	14	-1.683	0	-1.543	0	-0.938	✓✗	25	-0.728	6	-0.887	10	-1.790	✓✗ ⊕
	SEED – leak random	0	-1.301	0	-1.163	0	-1.430	✓✗	16	-0.770	13	-0.802	10	-3.100	✓✗ ⊕

Mode: Trojan insertion mode as in ISPD'23 contest; #(Vios): nr. of DRC and/or other design-check violations; TNS (ns): total negative slack; : Trojan insertion failed entirely, specifically due to exceedingly high utilization after insertion. Res: interpretation of final result, i.e., best-case outcome for adversaries across all three modes, as follows – ✗: Trojan insertion succeeded without any violations; ✓✗: Trojan insertion succeeded with managable violations / *TroLLoc* failed partially; ✓✓: Trojan insertion succeded but only with excessive violations / *TroLLoc* mainly successful. Note that *TroLLoc* interpretations are only applicable for protected layouts. Besides, note that, for cases where the outcomes are identical across different Trojan instances, only one row is listed, with the corresponding Trojans being provided in {} set notation. Finally note that, for cases where the final results' interpretation is the same for the baseline and protected layouts, we further annotate whether violations have ⊕ increased or ⊖ decreased for Trojan insertion into the protected layout over insertion into the baseline layout.

Results: We quantify the violation outcomes after Trojan insertion in Table V, thereby judging the effectiveness of *TroLLoc* versus the practicality of threats arising from layout-level Trojan insertion.[13] In Fig. 5, we provide some examples.

Overall, for the 9 Trojans in total, actual insertion was made more difficult in 7 case for *TroLLoc-MUX* and in 8 cases for *TroLLoc-XOR*, respectively, and made even impractical in 4

[13] Note the following for interpretation of Table V. Considering the number of DRC and/or other design-check violations, and the total negative slacks (TNS), we derive a conservative interpretation of the final result, i.e., the best-case outcome with least violations and smallest TNS value across all three different modes for Trojan insertion as in the ISPD'23 contest. The threshold between 'manageable' versus 'excessive' violations are, following some rules of thumb from the real-world of IC design, set to > 25 DRC violations and TNS values exceeding 20% of the clock period. For cases where both the baseline and the protected layouts lead to the same final result, we further compare the degree of violations, to judge whether *TroLLoc* managed to incur more or less challenges for the adversary toward final design closure after Trojan insertion. Toward that end, DRC violations are prioritized over TNS violations. This is because most if not all DRC violations must be fixed by adversary in order to release the design for actual manufacturing without raising suspicions or even failures, whereas *some* TNS violations may be kept, as these would 'merely' force the IC to only operate correctly for a reduced clock period—such outcome may well be blamed on process variations. Note that we focus here on TNS instead of WNS; TNS serves better to quantify the efforts imposed onto the adversaries for possibly fixing all timing issues, whereas WNS only quantifies by how much the whole IC violates timing.

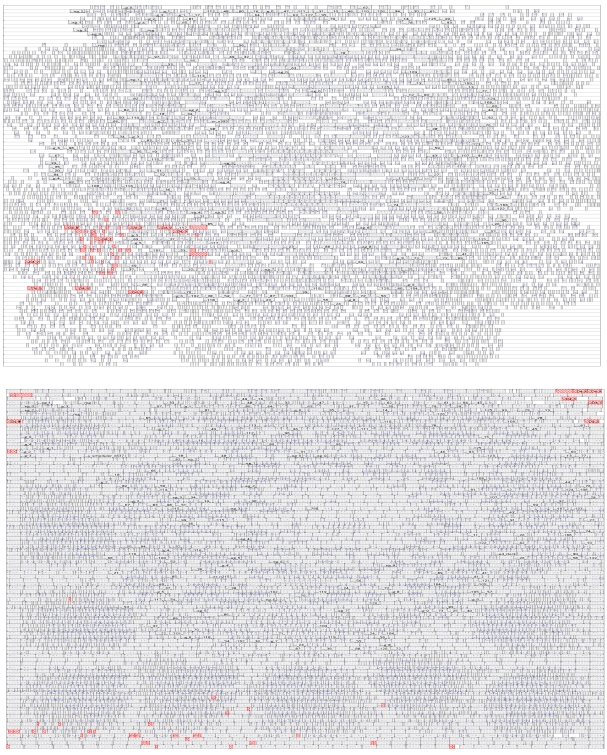

Fig. 5. Layout illustrations for the benchmark *Camellia*, after the Trojan *leak targeted* is inserted using the ISPD'23 'reg' mode. (Top) Trojan inserted into the baseline layout. (Bottom) Trojan inserted into the final layout as protected by *TroLLoc-MUX*. All benign cells, including *TroLLoc-MUX* instances, are marked in grey, whereas Trojan cells are marked in red. Note that layouts heights are reduced here; the original aspect ratio is 1.0.

and 6 out of those cases where excessive violations are induced on the attackers' efforts. More details are discussed next.

First, *TroLLoc* in general is particularly effective for *Camellia*, i.e., a benchmark where ultra-high utilization is achieved for the protected layouts. Some related layout examples are shown in Fig. 5: while Trojan insertion was successful without any violations for the baseline layout, it effectively failed due to excessive violations for the protected layout. For the latter, also compare to Fig. 4, to see how the Trojan logic only barely fitted into the few remaining open sites. Due to the resulting spread-out nature of the Trojan placement, excessive timing violations did arise; due to the ultra-high utilization, a large number of DRC violations did arise as well.

Second, *TroLLoc-XOR* is slightly more effective than *TroLLoc-MUX*. Interestingly, the layouts protected by *TroLLoc-XOR* exhibit little lower utilization numbers (Table IV). This implies that layout hardening against post-design insertion of actual Trojans is more intricate than 'only' requiring to push for ultra-high utilization. This finding argues the converse of earlier prior art like [36], [10], [34], but aligns with more recent and more practical works like [29].

Third, although *TroLLoc* cannot prevent Trojan insertion altogether, it is still effective. Here it is important to note that, while not directly comparable due to different technology setups, none of the winning teams of the ISPD'23 contest could fully prevent Trojan insertion either [11]. This implies that post-design insertion of actual Trojans, specifically through commercial-grade ECO techniques, represents a severe threat. This finding aligns with recent prior art [20], [28], [29].

Fourth, while not explicitly quantified here, it is important to note that, even in cases where Trojan insertion succeeded with manageable violations, the adversaries would still face two challenges: (i) to resolve the violations, which can become more challenging than usual for these ultra-dense layouts, and (ii) to be able to utilize the Trojan later on in the field. For (i), note that adversaries would have to tackle most if not all DRC violations and also keep TNS violations within tight bounds; also recall Footnote 13. For (ii), note that each and every security asset targeted at by Trojans has to be accounted for its underlying obfuscation, i.e., all the assets' signal values could be either as is or inverted, depending on the unknown locking key. Therefore, although not explicitly quantified here either, all the actual Trojan implementations would likely become more complex and larger and, thus, more difficult to insert.

To summarize and answer the research question raised in Sec. VI-B: adversaries are considerably but not entirely hindered by *TroLLoc* for post-design Trojan insertion, whereas an assessment of actual prospects requires case-by-case studies. From the perspective of commercial-grade IC design, this is expected: any SOTA IC design requires thorough efforts even for regular, security-agonistic research and development.

D. Case Study 3: Second-Order Attacks on TroLLoc-MUX

Here we evaluate *TroLLoc-MUX* against SOTA ML-based attacks that specifically target on MUX-based locking: *MuxLink* [16] and *SCOPE* [19]. We also utilize a SOTA structural attack, *Resynthesis* [18].

Setup for *MuxLink*: We obtain the *MuxLink* implementation from [16] and configure it as follows. First, we extend the one-hot feature vectors to capture all possible gates utilized in the ISPD'22 suite. Second, consistent with the original operation, we represent each gate, PI, and PO as a separate node in a graph, with all connections between gates and ports represented as edges. Note that the connections between the input/output of *TroLLoc-MUX* key-gates are kept as test set for prediction, while all others are used as training set. Finally, we adopt the same graph neural-network configuration and training hyperparameters as in [16]; specifically, the number of hops while extracting subgraphs is set to 3 and the threshold for post-processing is set to 0.01.

Setup for *SCOPE* and *Resynthesis* + *SCOPE*: We obtain the *SCOPE* implementation from [19] and the *Resynthesis* + *SCOPE* implementation from [73]. Note that *Resynthesis* employs majority voting for many *SCOPE* runs conducted over a large range of resynthesized netlists, which are structurally different yet functionally equivalent to the design under attack. Thus, we refer to this setup as *Resynthesis* + *SCOPE*.

We configure both tools as follows. First, since *SCOPE* cannot handle any loops in the designs, e.g., introduced through sequential feedback operations on FFs, we convert any FFs into pseudo PIs/POs. Note that this is the technically correct handling of sequential designs as unrolled combinational designs, and doing so is common practice in the literature. Also note that, for the FFs building up the shift register storing the

TABLE VI
RESULTS FOR *MuxLink* ON *TroLLoc-MUX* AND *D-MUX*

Design	*TroLLoc-MUX*				*D-MUX*			
	AC	*PC*	*KPA*	*#(X)*	*AC*	*PC*	*KPA*	*#(X)*
AES_1	28.7%	71.4%	50.2%	512	78.1%	79.9%	79.5%	21
AES_2	28.1%	75.0%	52.9%	564	80.4%	81.7%	81.4%	15
AES_3	08.1%	90.0%	45.0%	361	88.4%	89.1%	89.0%	3
Camellia	26.1%	73.8%	49.9%	606	90.6%	92.6%	92.4%	25
CAST	37.4%	66.0%	52.4%	449	93.7%	94.3%	94.2%	7
MISTY	33.3%	65.8%	49.3%	395	97.8%	98.0%	98.0%	3
openMSP430_1	23.5%	76.6%	50.1%	646	NA	NA	NA	NA
openMSP430_2	09.3%	90.4%	49.5%	440	66.8%	69.9%	69.0%	17
PRESENT	11.6%	95.0%	70.0%	201	NA	NA	NA	NA
SEED	37.0%	64.0%	50.7%	435	97.3%	97.7%	97.7%	6
SPARX	05.3%	94.9%	51.4%	2,312	NA	NA	NA	NA
TDEA	01.8%	99.0%	66.6%	208	NA	NA	NA	NA

AC: accuracy; PC: precision; KPA: key prediction accuracy; #(X): nr. of undeciphered key-bits; NA: locking using the script in [16] fails.

key, we designate all related pseudo PIs as individual key-bit PIs. Second, we utilize the *Resynthesis + SCOPE* framework with its default settings; the framework then automatically handles all required steps, like conversion from *Verilog* to *bench* netlists as required for *SCOPE*, etc.

Metrics: We are using the following established metrics: *accuracy (AC)*, *precision (PC)*, and *key prediction accuracy (KPA)*. 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., $(k_{correct}/(k_{total} - k_X))$. Furthermore, we consider the *constant propagation effect (COPE)*, which specifically measures the vulnerability against the *SCOPE* attack; in short, COPE = 0% means the attack fails entirely. All metrics are reported in percentage.

1) Results for *MuxLink*: The results in Table VI show that *MuxLink* can predict only 20.85% of all key-bits correctly, on average. Interestingly, it holds across the various benchmarks (except for *SPARX*) that, the larger is the key length, the higher is *MuxLink*'s accuracy. However, this does *no* imply that designs hardened with larger *TroLLoc-MUX* keys would be more prone to second-order attacks. This is not the case because, at the same time, (1) the average KPA is 53.17% which shows that, overall, *MuxLink* is not performing much better than random guessing, and (2) that average KPA remains rather steady across designs. In other words, for designs with larger *TroLLoc-MUX* keys, while *MuxLink* succeeds to correctly infer a larger share of key-bits, it also infers an equally larger share of key-bits incorrectly. Thus, the scalability of *TroLLoc-MUX*'s robustness against *MuxLink* is confirmed.

For the sake of putting *MuxLink*'s performance on *TroLLoc-MUX* into some context for prior art, we have implemented *D-MUX* [74], [16] and configured it for for the same key length as *TroLLoc-MUX* (Table IV). The corresponding results, also shown in Table VI, clearly demonstrate that *TroLLoc-MUX* is superior to *D-MUX*, across all metrics.

2) Results for *SCOPE*: The results in Table VII show that *SCOPE* can predict only 11.81% / 11.67%[14] of all key-bits correctly, on average. The average KPA of 52.79% / 47.20% shows that *SCOPE* is largely equivalent to random guessing as well. Note that we observe similar trends for key length and accuracy with *SCOPE* as with *MuxLink*; the scalability of *TroLLoc-MUX*'s robustness against *SCOPE* is confirmed. Also note that *SCOPE* can correctly decipher only a single bit for the benchmark TDEA, which trivially results in $PC(1) = KPA(1) = 100\%$ and $PC(2) \approx 100\%, KPA(2) = 0\%$.

3) Results for *Resynthesis + SCOPE*: The results in Table VII show that *Resynthesis + SCOPE* can predict 26.95% / 27.31% of all key-bits correctly, on average. While these accuracy results are superior to those of *SCOPE* alone—as expected—this does *not* mean that *TroLLoc-MUX* is more prone to second-order attacks by *Resynthesis + SCOPE*. This is because the average KPA of 49.16% / 50.83% shows that *Resynthesis + SCOPE* does not perform any better than random guessing either. Similar trends for key length and accuracy are observed again; the scalability of *TroLLoc-MUX*'s robustness against *Resynthesis + SCOPE* is confirmed as well.

[14] *SCOPE* predicts two mutually complementary keys. The related metrics are reported as 'AC(1)' versus 'AC(2)' etc., respectively.

E. Case Study 4: Second-Order Attacks on TroLLoc-XOR

Setup: Here we evaluate *TroLLoc-XOR* against *OMLA* [17], a SOTA attack that specifically targets on XOR-based locking. We obtain the *OMLA* implementation from [17]. Note that, unlike *SCOPE* or *Muxlink*, *OMLA* always provides an inference for all key-bits. Thus, $k_X = 0$ and $AC = PC = KPA$.

We configure this case study as follows. First, note that, in its original implementation, when given a locked design to attack, *OMLA* seeks to lock further parts of that design and then utilizes those newly locked parts as training data. For *TroLLoc-XOR*, however, any further locking is impractical, as the protected layouts are already locked as much as possible, exhibiting ultra-high utilization and/or very few relevant nets remain unlocked. Thus, for this case study, we train *OMLA* via a classical leave-one-out scheme, considering all respectively remaining, locked designs from the ISPD'22 suite.

Second, based on the findings for *TroLLoc-XOR* without gate balancing (Sec. VI-E1), we revise the cell selection described in Algorithm 1 as follows. For each gate type, we track the number of locked instances. During the iterative selection and locking procedure, we then balance each type with its complementary counterpart, e.g., `NAND2` and `AND2`. Before each round of cell selection, we temporarily ignore a cell if locking that cell would break the balance.

1) Results for *TroLLoc-XOR* without Gate Balancing: The results in Table VIII show that, without gate balancing being applied for *TroLLoc-XOR*, *OMLA* can achieve an average KPA of 69%, which is notably better than random guessing. This is due to the fact that the composition of locked gates can be unbalanced in terms of the original, unlocked gate's type. For example, in *AES_1*, 441 `NAND2` gates are locked while there are only 37 `AND2` gates locked for *TroLLoc-XOR* without balancing applied. Such trends occur in other designs as well, from which *OMLA* learns about the majority of the original gates' types. That is, for the same example of *AES_1*, whenever *OMLA* seeks to decipher a *TroLLoc-XOR* key-gate that is locked via `NAND2` or `AND2` gates, it will predict the original gate to be a `NAND2` with much higher probability.

TABLE VII
RESULTS FOR *SCOPE* AND *Resynthesis + SCOPE* ON *TroLLoc-MUX*

Design	***SCOPE***							***Resynthesis + SCOPE***						
	AC(1)	*PC(1)*	*KPA(1)*	*AC(2)*	*PC(2)*	*KPA(2)*	*COPE*	*AC(1)*	*PC(1)*	*KPA(1)*	*AC(2)*	*PC(2)*	*KPA(2)*	*Avg. COPE*
AES_1	21.10%	79.23%	50.39%	20.76%	78.89%	49.60%	18.45%	32.77%	68.14%	50.70%	31.85%	67.22%	49.29%	17.89%
AES_2	22.46%	78.03%	50.56%	21.96%	77.53%	49.43%	17.87%	33.94%	65.97%	49.93%	34.02%	66.05%	50.06%	17.13%
AES_3	9.52%	90.92%	51.21%	9.07%	90.47%	48.78%	10.32%	12.24%	86.16%	46.95%	13.83%	87.75%	53.04%	9.96%
Camellia	11.40%	91.42%	57.08%	8.57%	88.59%	42.91%	0.10%	34.46%	68.13%	51.95%	31.86%	65.53%	48.04%	0.23%
CAST	12.40%	87.91%	50.64%	12.08%	87.59%	49.35%	0.05%	41.92%	58.96%	50.53%	41.03%	58.07%	49.46%	0.17%
MISTY	17.77%	81.15%	48.53%	18.84%	82.22%	51.46%	0.35%	40.49%	57.94%	49.05%	42.05%	59.50%	50.94%	0.53%
openMSP430_1	16.09%	84.07%	50.25%	15.92%	83.90%	49.74%	0.24%	27.99%	72.41%	50.36%	27.58%	72.00%	49.63%	0.65%
openMSP430_2	13.25%	88.76%	54.13%	11.23%	86.74%	45.86%	0.18%	23.02%	81.03%	54.82%	18.96%	76.97%	45.17%	0.95%
PRESENT	1.24%	94.60%	18.75%	5.39%	98.75%	81.25%	2.79%	18.25%	75.51%	42.71%	24.48%	81.74%	57.28%	2.40%
SEED	11.97%	87.22%	48.37%	12.77%	88.02%	51.62%	0.05%	40.63%	58.62%	49.54%	41.37%	59.36%	50.45%	0.16%
SPARX	4.02%	96.51%	53.60%	3.48%	95.97%	46.39%	0.14%	7.39%	92.40%	49.35%	7.59%	92.60%	50.64%	0.16%
TDEA	0.46%	100%	100%	0.00%	99.53%	0.00%	0.01%	10.28%	86.91%	44.00%	13.08%	89.71%	56.00%	0.60%

AC: accuracy; PC: precision; KPA: key prediction accuracy; COPE: constant propagation effect [19].

TABLE VIII
RESULTS FOR *OMLA* ON *TroLLoc-XOR*

Design	***TroLLoc-XOR*, Unbalanced**			***TroLLoc-XOR*, Balanced**		
	KL	*Utils*	*KPA*	*KL*	*Utils*	*KPA*
AES_1	1,200	96.1%	79.1%	1,273	95.0%	51.5%
AES_2	1,149	95.0%	78.9%	1,215	94.3%	52.1%
AES_3	416	95.5%	63.7%	416	95.6%	52.6%
Camellia	1,348	98.7%	75.6%	1,051	86.8%	48.8%
CAST	1,804	94.6%	78.2%	1,707	96.2%	68.4%
MISTY	1,315	93.9%	78.9%	1,172	93.8%	67.6%
openMSP430_1	1,337	98.7%	70.2%	1,154	91.2%	56.8%
openMSP430_2	582	97.4%	66.5%	582	97.2%	55.0%
PRESENT	255	97.4%	47.8%	208	88.7%	52.9%
SEED	1,802	94.1%	78.5%	1,618	92.4%	66.9%
SPARX	2,746	99.0%	56.6%	2,741	98.9%	48.2%
TDEA	226	99.0%	54.0%	226	99.0%	53.5%

KL: key length (bits), accounting both for locked assets and other locked cell instances; Utils: utilization after physical synthesis; KPA: key prediction accuracy.

2) Results for *TroLLoc-XOR* with Gate Balancing: The results in Table VIII show that, with gate balancing being applied for *TroLLoc-XOR*, *OMLA* can only achieve an average KPA of 56.19%, which is much closer to random guessing than *TroLLoc-XOR* without gate balancing, namely by 12.81 percentage points (pps). For the previous example of *AES_1*, KPA is even reduced by 27.60 pps.

We note that *TroLLoc-XOR* with gate balancing applied can be limited—by the original gate composition—in terms of achievable utilization and key length. In fact, key lengths are 5.76% shorter and utilization numbers are 2.53 pps lower on average. To mitigate this limitation, guided efforts are required from the IC-design stage prior to physical synthesis, i.e., logic synthesis;[15] such efforts are left for future work.

VII. CONCLUSION

We propose a novel scheme for proactive, pre-silicon Trojan prevention by means of logic locking and layout hardening, all in a real-world setup for IC design. Our work successfully addresses three key challenges for related prior art and for Trojan countermeasures in general: (i) ours remains robust against potential second-order attacks, i.e., adversaries seeking to bypass the layout-level defense before actual Trojan insertion, (ii) ours is effective as in protecting layouts in general and security assets in particular against real-world insertion of various Trojans, and (iii) ours is efficient as in achieving favorable security-versus-cost trade-offs, by careful integration of our techniques into commercial-grade IC tools.

[15]Gate balancing could become another optimization objective, in addition to the classical power, performance, and area (PPA) objectives. Similar efforts for security-aware logic synthesis already exist, e.g., to reduce power side-channel information leakage by means of synthesizing differential logic [6].

REFERENCES

[1] F. Wang *et al.*, "Security closure of IC layouts against hardware Trojans," in *Proc. ISPD*, 2023.
[2] ——, "Baseline and protected layouts," 2024, https://drive.google.com/file/d/17ysAYnoPgFuy0Yj969dTGUWZVrIT7PmW/view?usp=sharing.
[3] M. Rostami *et al.*, "A primer on hardware security: Models, methods, and metrics," *Proc. IEEE*, vol. 102, no. 8, 2014.
[4] W. Hu *et al.*, "An overview of hardware security and trust: Threats, countermeasures and design tools," *TCAD*, 2020.
[5] J. Knechtel, "Hardware security for and beyond CMOS technology," in *Proc. ISPD*, 2021.
[6] J. Knechtel *et al.*, "Towards secure composition of integrated circuits and electronic systems: On the role of EDA," in *Proc. DATE*, 2020.
[7] G. Sigl, "Keynote address: Design of secure systems – where are the eda tools?" in *Proc. ICCAD*, 2011.
[8] P. Ravi *et al.*, "Security is an architectural design constraint," *Microprocess. Microsyst.*, vol. 68, no. C, 2019.
[9] J. Knechtel *et al.*, "Benchmarking security closure of physical layouts: Ispd 2022 contest," in *Proc. ISPD*, 2022.
[10] ——, "Security closure of physical layouts," in *Proc. ICCAD*, 2021.
[11] M. Eslami *et al.*, "Benchmarking advanced security closure of physical layouts: Ispd 2023 contest," 2023, https://wp.nyu.edu/ispd23_contest/.
[12] K. Xiao *et al.*, "Hardware trojans: Lessons learned after one decade of research," *TODAES*, vol. 22, no. 1, 2016.
[13] C. Dong *et al.*, "Hardware trojans in chips: A survey for detection and prevention," *Sensors*, vol. 20, no. 18, 2020.
[14] K. Yang *et al.*, "A2: Analog malicious hardware," in *Proc. SP*, 2016.
[15] M. Yasin *et al.*, *Trustworthy Hardware Design: Combinational Logic Locking Techniques*. Springer, 2020.
[16] L. Alrahis *et al.*, "Muxlink: Circumventing learning-resilient mux-locking using graph neural network-based link prediction," in *Proc. DATE*, 2022.
[17] ——, "Omla: An oracle-less machine learning-based attack on logic locking," *TCS*, vol. 69, no. 3, 2022.
[18] F. Almeida *et al.*, "Resynthesis-based attacks against logic locking," in *Proc. ISQED*, 2023.
[19] A. Alaql *et al.*, "Scope: Synthesis-based constant propagation attack on logic locking," *Trans. VLSI*, vol. 29, no. 8, 2021.
[20] T. Perez and S. Pagliarini, "Hardware Trojan insertion in finalized layouts: From methodology to a silicon demonstration," *TCAD*, 2022.
[21] S. Adee, "The hunt for the kill switch," *Spectrum*, vol. 45, no. 5, 2008.
[22] S. Skorobogatov and C. Woods, "Breakthrough silicon scanning discovers backdoor in military chip," in *Proc. TCHES*, 2012.
[23] M. Muehlberghuber *et al.*, "Red team vs. blue team hardware Trojan analysis: Detection of a hardware Trojan on an actual ASIC," in *Proc. Int. Workshop Hardw. Arch. Supp. Sec. Priv.*, 2013.

[24] S. Ghandali *et al.*, "Side-channel hardware Trojan for provably-secure SCA-protected implementations," *Trans. VLSI*, vol. 28, no. 6, pp. 1435–1448, 2020.
[25] E. Puschner *et al.*, "Red team vs. blue team: A real-world hardware Trojan detection case study across four modern CMOS technology generations," in *Proc. SP*, 2023.
[26] T. Trippel *et al.*, "Icas: an extensible framework for estimating the susceptibility of ic layouts to additive trojans," in *Proc. SP*, 2020.
[27] H. Salmani *et al.*, "Vulnerability analysis of a circuit layout to hardware trojan insertion," *Trans. Inf. Forens. Sec.*, vol. 11, no. 6, 2016.
[28] A. Hepp *et al.*, "A pragmatic methodology for blind hardware Trojan insertion in finalized layouts," in *Proc. ICCAD*, 2022.
[29] X. Wei *et al.*, "Rethinking ic layout vulnerability: Simulation-based hardware trojan threat assessment with high fidelity," in *Proc. SP*, 2024.
[30] S. Dupuis *et al.*, "A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans," in *Proc. IOLTS*.
[31] A. Marcelli *et al.*, "An evolutionary approach to hardware encryption and trojan-horse mitigation," in *Proc. DATE*, 2017.
[32] M. S. Samimi *et al.*, "Hardware enlightening: No where to hide your hardware trojans!" in *Proc. IOLTS*, 2016.
[33] D. Šišejković *et al.*, "Control-lock: Securing processor cores against software-controlled hardware trojans," in *Proc. GLSVLSI*, 2019.
[34] K. Xiao *et al.*, "A novel built-in self-authentication technique to prevent inserting hardware trojans," *TCAD*, vol. 33, no. 12, 2014.
[35] P.-S. Ba *et al.*, "Hardware trojan prevention using layout-level design approach," in *Proc. Europ, Conf. Circ. Theory Des.*, 2015.
[36] ——, "Hardware trust through layout filling: A hardware trojan prevention technique," in *Proc. ISVLSI*, 2016.
[37] H. Hossein-Talaee and A. Jahanian, "Layout vulnerability reduction against trojan insertion using security-aware white space distribution," in *Proc. ISVLSI*, 2017.
[38] F. Imeson *et al.*, "Securing computer hardware using 3D integrated circuit (IC) technology and split manufacturing for obfuscation," in *Proc. USENIX Sec. Symp.*, 2013.
[39] Q. Shi *et al.*, "Obfuscated built-in self-authentication with secure and efficient wire-lifting," *TCAD*, vol. 38, no. 11, 2018.
[40] T. Trippel *et al.*, "T-TER: Defeating A2 Trojans with targeted tamper-evident routing," in *Proc. CCS*, 2023.
[41] G. Guo *et al.*, "ASSURER: A PPA-friendly security closure framework for physical design," in *Proc. ASP-DAC*, 2023.
[42] J.-W. Hsu *et al.*, "Security-aware physical design against Trojan insertion, frontside probing, and fault injection attacks," in *Proc. ISPD*, 2023.
[43] M. Eslami *et al.*, "SALSy: Security-aware layout synthesis," 2023.
[44] X. Wei *et al.*, "Gdsii-guard: Eco anti-trojan optimization with exploratory timing-security trade-offs," in *Proc. DAC*, 2023.
[45] M. Eslami *et al.*, "SCARF: Securing chips with a robust framework against fabrication-time hardware trojans," 2024.
[46] L. A. Guimarães *et al.*, "Detection of layout-level trojans by monitoring substrate with preexisting built-in sensors," in *Proc. ISVLSI*, 2017.
[47] Y. Hou *et al.*, "R2d2: Runtime reassurance and detection of a2 trojan," in *Proc. HOST*, 2018.
[48] A. Vijayan *et al.*, "Runtime identification of hardware trojans by feature analysis on gate-level unstructured data and anomaly detection," *TODAES*, vol. 25, no. 4, 2020.
[49] L. W. Kim and J. D. Villasenor, "A system-on-chip bus architecture for thwarting integrated circuit trojan horses," *Trans. VLSI*, 2011.
[50] A. Basak *et al.*, "Security assurance for system-on-chip designs with untrusted IPs," *Trans. Inf. Forens. Sec.*, vol. 12, no. 7, 2017.
[51] T. F. Wu *et al.*, "TPAD: Hardware trojan prevention and detection for trusted integrated circuits," *TCAD*, vol. 35, no. 4, 2016.
[52] R. S. Wahby *et al.*, "Verifiable ASICs," *Proc. SP*, 2016.
[53] D. Agrawal *et al.*, "Trojan detection using IC fingerprinting," in *Proc. SP*, 2007.
[54] X. Guo *et al.*, "QIF-Verilog: Quantitative information-flow based hardware description languages for pre-silicon security assessment," in *Proc. HOST*, 2019.
[55] N. Fern *et al.*, "Detecting hardware trojans in unspecified functionality through solving satisfiability problems," in *Proc. ASP-DAC*, 2017.
[56] X. Chen *et al.*, "Hardware trojan detection in third-party digital intellectual property cores by multilevel feature analysis," *TCAD*, 2018.
[57] H. Lashen *et al.*, "TrojanSAINT: Gate-level netlist sampling-based inductive learning for hardware Trojan detection," in *Proc. ISCAS*, 2023.
[58] S. Narasimhan *et al.*, "Hardware trojan detection by multiple-parameter side-channel analysis," *Trans. Comp.*, vol. 62, no. 11, 2013.
[59] D. Deng *et al.*, "Novel design strategy toward a2 trojan detection based on built-in acceleration structure," *TCAD*, vol. 39, no. 12, 2020.
[60] T. Sugawara *et al.*, "Reversing stealthy dopant-level circuits," *J. Cryptogr. Eng.*, vol. 5, no. 2, 2015.
[61] N. Vashistha *et al.*, "Trojan scanner: Detecting hardware trojans with rapid SEM imaging combined with image processing and machine learning," in *ISTFA*, 2018.
[62] V. Gohil *et al.*, "ATTRITION: Attacking static hardware Trojan detection techniques using reinforcement learning," in *Proc. CCS*, 2022.
[63] K. Hasegawa *et al.*, "Trojan-feature extraction at gate-level netlists and its application to hardware-Trojan detection using random forest classifier," in *Proc. ISCAS*, 2017.
[64] N. Jacob *et al.*, "Hardware Trojans: Current challenges and approaches," *Comp. & Dig. Techs.*, vol. 8, no. 6, 2014.
[65] K. Baek *et al.*, "Pin accessibility and routing congestion aware DRC hotspot prediction using graph neural network and u-net," in *Proc. ICCAD*, 2022.
[66] J. A. Roy *et al.*, "Ending piracy of integrated circuits," *Computer*, vol. 43, no. 10, 2010.
[67] H. Mardani Kamali *et al.*, "Lut-lock: A novel lut-based logic obfuscation for fpga-bitstream and asic-hardware protection," in *Proc. ISVLSI*, 2018.
[68] J. Rajendran *et al.*, "Fault analysis-based logic encryption," *Trans. Comp.*, vol. 64, no. 2, 2015.
[69] M. Yasin *et al.*, "Activation of logic encrypted chips: Pre-test or post-test?" in *Proc. DATE*, 2016.
[70] P. Subramanyan *et al.*, "Evaluating the security of logic encryption algorithms," in *Proc. HOST*, 2015.
[71] P. Chakraborty *et al.*, "Sail: Machine learning guided structural analysis attack on hardware obfuscation," in *Proc. AHOST*, 2018.
[72] "NanGate FreePDK45 Open Cell Library," Nangate Inc, 2011, http://www.nangate.com/?page_id=2325.
[73] J. Knechtel, "Resynthesis + SCOPE Setup," 2023, https://github.com/DfX-NYUAD/resynthesis_attack.
[74] D. Šišejković *et al.*, "Deceptive logic locking for hardware integrity protection against machine learning attacks," *TCAD*, vol. 41, no. 6, 2022.