Logic Encryption: This Time for Real

Rupesh Raj Karn[‡], Lakshmi Likhitha Mankali[†], Zeng Wang[†], Saideep Sreekumar[‡], Prithwish Basu Roy[†‡], Ozgur Sinanoglu[‡], Lilas Alrahis[§], Johann Knechtel[‡]

[†]NYU Tandon School of Engineering, New York, USA
[‡]NYU Abu Dhabi, Abu Dhabi, UAE
[§]Computer and Information Engineering, Khalifa University, Abu Dhabi, UAE

{rupesh.k, likhitha.mankali, zw3464, sds710, pb2718}@nyu.edu
ozgursin@nyu.edu, lilas.malrahis@ku.ac.ae, johann@nyu.edu

Abstract

Modern circuits face various threats like reverse engineering, theft of intellectual property (IP), side-channel attacks, etc. Here, we present a novel approach for IP protection based on logic encryption (LE). Unlike established schemes for logic locking, our work obfuscates the circuit's structure and functionality by encoding and encrypting the logic itself. We devise an end-to-end method for practical LE implementation based on standard cryptographic algorithms, key-bit randomization, simple circuit design techniques, and system-level synthesis operations, all in a correct-by-construction manner. Our extensive analysis demonstrates the remarkable efficacy of our scheme, outperforming prior art against a range of oracle-less attacks covering crucial threat vectors, all with lower design overheads. We provide a full open-source release.

Keywords

Hardware Security, IP Protection, Logic Locking, Logic Encryption, Oracle-Less Attacks, CAD Method

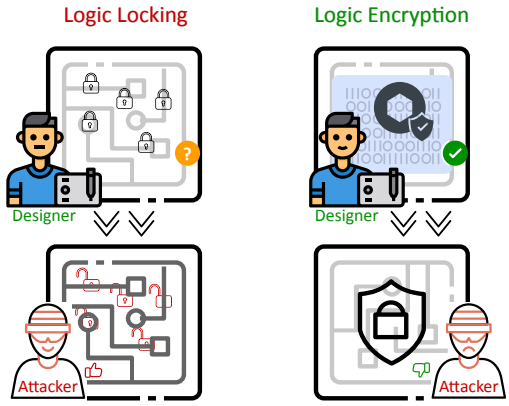

Figure 1: Our approach for revisiting LE is fundamentally different from prior art of LL. Instead of integrating KG structures into the netlist as an afterthought, our work encrypts the netlist itself, utilizing cryptographic algorithms and simple circuit design techniques, all to fully obfuscate the design IP with competitive overheads.

1 Introduction

Protecting integrated circuits (ICs) against intellectual property (IP) piracy and reverse engineering (RE), amongst other threats, has become increasingly vital in today's distributed landscape for design and manufacturing [1–5].

The term *logic encryption (LE)* was originally coined for various techniques that embed additional key-gate (KG) structures into IC designs [6–8]; correct operation of the IC requires the correct key-bit for these structures. While those early works established a solid foundation for protection of modern IC designs, unlike the wording suggests, they did not pursue actual encryption of the logic.

Nowadays, this approach is more commonly known as *logic locking (LL)*. By integrating KG structures within the netlist more as an afterthought, LL can fall short in truly protecting the design IP. In fact, numerous attacks have shown, time and again, significant vulnerabilities across LL schemes, e.g., [5, 9–24]. Our work signifies a paradigm shift, back to the roots of the wording. For the first time, we advocate for encryption of the logic itself, aiming for full-scale obfuscation of the design IP. Our approach to LE involves encoding the original circuit into binary plaintexts, which are encrypted using established cryptographic algorithms. The resulting ciphertexts are decoded back into an encrypted circuit. The latter is integrated with some decryption-like circuitry into the final protected design. Full details for our method are provided in Sec. 3 and our release [25]. We illustrate the high-level difference between established LL techniques and our approach to LE in Fig. 1.

In short, our paper provides several contributions:

(1) *Logic Encryption*: Revisiting the foundation of logic encryption/locking, our approach emphasizes on encrypting the logic itself. Despite the long history of the field, this is a first-of-its-kind work. The motivation is to fully obfuscate the design IP against advanced attacks.
(2) *End-to-End, Commercial-Grade, Open-Source Method*: We devise and implement a comprehensive end-to-end method for realizing LE. Our design analysis, and our method as a whole, utilize commercial-grade CAD tools, rendering it robust and relevant for real-world insights. We provide a full open-source release at [25].
(3) *Thorough Security and Design Evaluation*: We conduct an extensive evaluation of our approach for both security and power, performance, and area (PPA). We also compare ours to state-of-the-art (SOTA) for LL [26–28]. Our security analysis is based on open-source attack frameworks and covers critical threat vectors for the oracle-less setting:
   (a) machine learning (ML)-based prediction of KG structures [15],
   (b) ML-based prediction of KG interconnects [16],

(c) resynthesis-based exploitation of information leakage by KG structures and interconnects [14, 17],
(d) and ML-based RE [29].

2 Preliminaries

We begin with a concise overview of several key attacks—**OMLA [15]**, **TRLL [26]**, **gDMUX [27]**, **LUT-L [30]**, MuxLink [16], GNN-RE [29] and the resynthesis attack [14]. This foundation will facilitate the reader's comprehension before we present the implementation of these attacks on our logic encryption scheme in Sections 4.2 to 4.6.

A plethora of LL schemes exist, e.g., [5, 26–28, 31–36], but most have been challenged over time [5, 9–20, 23, 24, 37]. While early attacks followed the oracle-guided threat model, e.g., [38, 39], the recent rise of oracle-less attacks now dominates [14–17, 40–43] Thus, in this work, we also focus on the oracle-less threat model.

In traditional LL with XOR/XNOR KGs, information leakage arises from the KG type directly mapping to the correct key-bit: 0 for XOR and 1 for XNOR [6]. Logic synthesis is employed after KG insertion in an effort to disrupt this mapping. Still, ML-based attacks like OMLA [15] and resynthesis attack [14] show that such synthesis efforts (i) are deterministic, with predictable circuit modifications depending on the key-bits, and (ii) their effects often being localized.

Advanced LL Schemes: Based on these insights, researchers strive for LL schemes that do not rely on synthesis for resilience. Representative schemes are outlined next.

Truly random LL (TRLL) [26] performs its own bubble pushing and inverter absorption to insert XOR/XNOR KGs, aiming for a truly randomized correlation between key-bits and KG types.

LL may utilize multiplexers (MUXes) for their inherent structural ambiguity [27, 33]. For example, a 2-to-1 MUX acting as KG takes two data inputs, a true wire and a false wire, and a key-input drives the select line. As the true wire can be randomly connected either to the first or second data input of the MUX, the correct key-bit is truly random 0 or 1, with no specific key-bit associated with the MUX KG as such. The scheme gDMUX [27] goes further by strategically constructing pairs of MUX KGs such that the circuit remains fully connected regardless of the key-bit applied. LL may also utilize key-controlled look-up tables (LUT) to redact gates or whole blocks from the circuit [30];

we refer to this scheme as LUT-L in the remainder.

Another kind of redaction is proposed in [44]: their idea is to embed circuits into cryptographic hardware primitives, thereby locking the circuits' IP, but not encrypting it.[1]

Advanced Attacks: Despite their promises, these LL schemes remain vulnerable. Representative attacks are outlined next.

SCOPE [17] explores both possible key-bit values, 0 vs. 1, for each KG structure and, for each possible combination of key-bit values across all KG structures, runs simple resynthesis efforts. It then infers any correlation between the key-bit values and the resulting PPA features, e.g., smaller area due to removal of redundant logic induced by incorrect key-bit values.

The resynthesis attack [14] advances this approach, by first running synthesis for various settings, including challenging timing constraints on KG structures, and subsequently providing SCOPE with many more structural variations to explore. Recently, this approach was successfully extended to compound LL schemes,[2] realizing a divide-and-conquer strategy, as in first separating the different LL schemes and then tackling them separately [43].

For TRLL, while KG structures are better obfuscated, more intricate circuit characteristics, such as the number of inverters and the overall distribution of gate types, still allowed advanced ML-based attacks to succeed [42].

MuxLink [16] is tailored for MUX-based locking; it considers the challenge of deciphering the correct key-bit as a link-prediction problem. MuxLink achieves accuracy as high as 100% on gDMUX and other SOTA schemes. **Reverse Engineering:** Another important aspect, which is often overlooked, is whether LL can also prevent RE. Note that RE is less stringent than key-recovery attacks; RE aims to understand the overall functionality of the design, not necessarily all key-bits and full circuit details [29, 46].

GNN-RE [29] provides a modern approach to RE; it trains a graph neural network (GNN) on a circuit dataset with structural variations, e.g., comprising different adder implementations, to identify the functionality of unseen IP. Notably, GNN-RE is able to identify IP even with LL in place [29].

3 Method for Logic Encryption

The fully automated, end-to-end method for LE is outlined in Fig. 2. Next, we provide details for each stage. Note that implementation examples are provided in our release at [25]. Furthermore note that, while we utilize synthesis throughout all stages, this is only for implementation efforts—the resilience of our scheme does not depend on synthesis obfuscating KG structures. We also demonstrate this in our security analysis in Sec. 4.

3.1 Construction of Encrypted Circuit

The first stage, labelled Ⓐ in Fig. 2, is the construction of the encrypted circuit (EC). The EC is a core component for LE, along the correction circuit (Sec. 3.2).

1) Devise Coding Scheme: First, the original circuit (OC) is prepared for encryption. We must encode all gates such that subsequent decoding (after encryption) is complete and deterministic. Thus, we require a coding scheme where the set of code-words fully covers the set of gate types. For simplicity, we also require that all gate types have the same number of inputs/outputs, to maintain the original interconnects as such.

In this work, for simplicity but without loss of generality (w/o.l.o.g.), we utilize the set of universal NAND and NOR gates. Importantly, this simple coding scheme does not undermine the subsequent cryptography-enabled obfuscation of the OC. The actual coding scheme, i.e., the assignment of code-words 0 and 1 to NAND and NOR, or vice versa, is randomly decided and memorized.

[1] Their approach is orthogonal to ours for multiple reasons. (1) They embed circuits into cryptographic hardware, whereas we utilize cryptographic algorithms for actual circuit/logic encryption. (2) Their applicability is severely limited: less than 10 gates are redacted across all experiments in [44], yet with significant PPA overheads. (3) Theirs was proposed exclusively for the oracle-guided threat model. In that context, another attack [45] questioned their security against power side-channel attacks.

[2] The LL schemes described above aim for resilience in the oracle-less threat model, but not the oracle-guided threat model, whereas other schemes like [34, 44], often referred to as "provably secure LL", aim for the reverse. Compound locking integrates schemes from both worlds, aiming for more holistic defenses [1, 19],

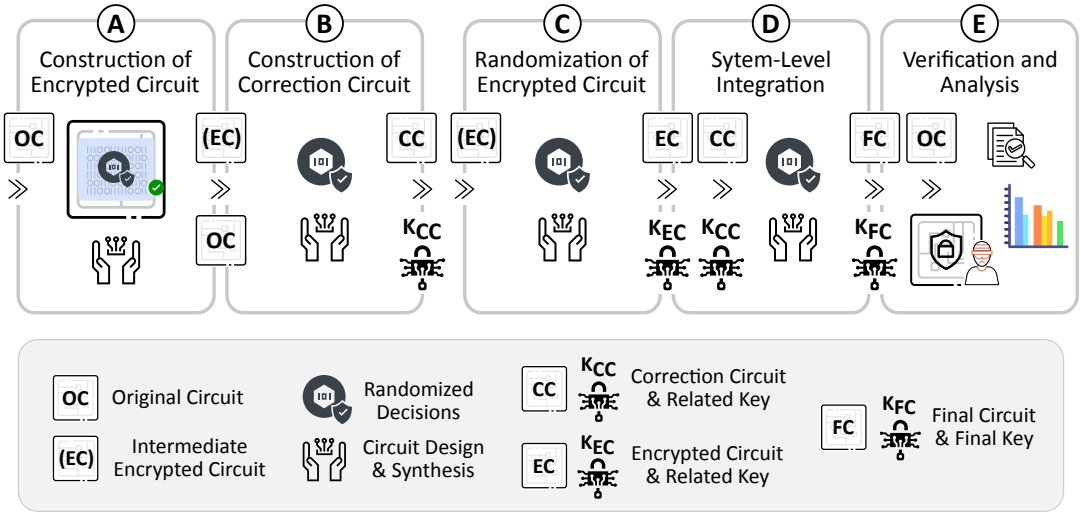

Figure 2: End-to-end method for LE, separated into key stages.

2) Resynthesize OC and Encoding of Gates: Once the coding scheme is finalized, the OC is resynthesized using only the gate types supported in the scheme. Then, each gate in the resynthesized OC is encoded. The order in which all gates are selected during encoding is randomly decided and memorized. The result of this step is a binary plaintext, of length g bits (in this work) for g gates in the resynthesized OC.

3) Encryption of Encoded Gates: The plaintext, i.e., the encoded representation of the resynthesized OC, is encrypted. W/o.l.o.g., we utilize AES-128 with random keys.[3] The plaintext of g bits in total is split into 128-bit messages, and the last message is padded with random bits as needed. The result of this step is a binary ciphertext, representing the EC.

This step ensures that attackers cannot decipher the true functionality of the OC from the EC without the encryption key. Based on the well-known cryptographic principles of confusion and diffusion, any correlations between the logic of the OC and EC are truly obfuscated. In short, we utilize cryptography for truly randomized obfuscation of the entire OC and all its IP, which is a major difference over prior art in LL.

4) Decoding of Gates into EC: Utilizing the coding scheme, the ciphertext, and the memorized order of gates selected during encoding, all gates are decoded. That is, for every gate g_i, the corresponding bit c_i of the ciphertext is decoded into either a NAND or NOR gate. The result of this step is the basic EC.

5) Resynthesize EC: To maintain PPA efficiency, the basic EC is resynthesized using the full library. Importantly, this step does not undermine the functional-centric circuit transformations realized by encryption, as resynthesis guarantees functional equivalence by construction. The result of this step is the intermediate EC, as required for the next stage (Sec. 3.2), whereas the final EC is obtained only later (Sec. 3.3).

[3]We do not memorize these keys, as we do not decrypt the EC later on; see Sec. 3.2.

3.2 Construction of Correction Circuit

Next, labelled Ⓑ in Fig. 2, we construct the correction circuit (CC). The CC ensures that the final circuit (FC) can realize the intended functionality despite the encrypted logic. Importantly, the CC is *not* constructed by decryption of the EC; rather, the CC is constructed such that it operates on the primary outputs (POs) of the EC for correction of the functional behaviour.

1) Devise CC Logic, Including Random Keys: Let PO_{OC} denote a PO of the OC and PO_{EC} the corresponding PO of the EC. Then, the logic for the corresponding PO of the CC is obtained as

$$PO_{CC} = PO_{OC} \oplus PO'_{EC}$$

where $\oplus$ denotes the XOR operation and PO'_{EC} is obtained from PO_{EC} in a randomized manner, either as-is (buffered) or inverted. Importantly, these random choices are memorized for all POs, as so-called intermediate key-bits K_{CC}, whose value is 0 for buffering / 1 for inverting the logic, respectively.

2) Synthesize CC: For all POs, the above logic is compiled into a top-level *Verilog* module, also called wrapper. The wrapper is synthesized with the OC and the intermediate EC as instantiated modules. The result of this step is the final CC.

3.3 Randomization of Encrypted Circuit

The third stage, labelled as Ⓒ in Fig. 2, is to randomize the EC. The motivation here is as above, i.e., to obtain some further key-bits essential for correct operation of the FC.

1) Devise EC Logic, Including Random Keys: Let PO_{EC} denote a PO of the intermediate EC. Then, its reassignment can be expressed as $PO_{EC,\text{new}} = PO'_{EC}$ where PO'_{EC} is obtained from PO_{EC} in a randomized manner, either as-is (buffered) or inverted. As before, these random choices are memorized, as intermediate key-bits K_{EC} of value 0 for buffering / 1 for inverting the logic, respectively.

2) Synthesize Final EC: For all POs, the above logic is compiled into another wrapper, which is synthesized with the intermediate EC as module. The result of this step is the final EC.

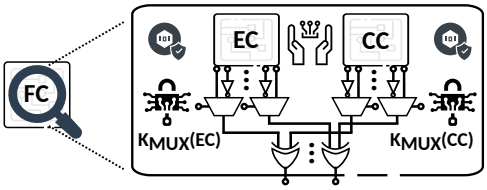

Figure 3: Obfuscation of system-level interconnects between the encrypted circuit and the correction circuit.

3.4 System Integration

The fourth stage, labelled as Ⓓ in Fig. 2, is the secure system-level integration of the EC and the CC into the FC. Toward that end, we utilize MUX-based KG structures that introduce yet another layer of obfuscation, namely for the correct connections between the EC and the CC. These KG structures are shown in Fig. 3 and explained in detail next. Importantly, our use of KG structures here differs from prior art as they are not introduced throughout the netlist as afterthought, but rather as key components at the system level.

1) Devise FC Logic, Including Random Keys: We feed any pair of corresponding POs from the EC and the CC to individual XOR gates, which in turn drive the FC's POs. This enables restoration of the intended functionality as follows:

$$PO_{FC} = PO_{EC} \oplus PO_{CC} \tag{1}$$
$$= PO_{EC} \oplus (PO_{OC} \oplus PO_{EC}) = PO_{OC} \tag{2}$$

Note that we are not feeding the EC/CC POs directly to XOR gates, but first pass them to the MUX KG structures. More specifically, for both EC and CC POs, we randomly and independently assign the respective buffered/inverted PO logic to the two data inputs of these KG structures. These randomized decisions are memorized as intermediate key-bits K_{MUX}, which are of value 0 for the buffered PO logic connected to the first data input and the inverted PO logic to the second data input, and of value 1 for the vice-versa assignment. Importantly, the correct key-bit values could be either 0 or 1 here, both depending on these randomized decisions for the MUX KG construction as well as the previously made randomized decisions to buffer or invert the EC/CC POs during their construction. In simpler words, the different intermediate key-bits are entangled by construction.

2) Derive Final Keys: Altogether, we have three sets of intermediate key-bits, K_{EC}, K_{CC}, and K_{MUX}. Note that, while K_{EC} and K_{CC} are of size $|PO|$, K_{MUX} and K_{FC} are of size $2 \times |PO|$, with $|PO|$ being the same across OC, EC, CC, and FC. The final key-bits, K_{FC}, are independently derived for each PO as follows, considering the entanglement of the intermediate key-bits:

$$K_{FC}(PO_{EC}) = K_{MUX}(PO_{EC}) \oplus K_{EC}(PO) \tag{3}$$
$$K_{FC}(PO_{CC}) = K_{MUX}(PO_{CC}) \oplus K_{CC}(PO) \tag{4}$$

3) Synthesize FC: For all POs, the above logic for EC and CC integration with MUX KG structures and XOR assignments is compiled into yet another wrapper. The wrapper is synthesized with the final EC and the final CC as instantiated modules. The result of this step is the final circuit, FC.

3.5 Verification and Analysis

The final stage, labelled as Ⓔ in Fig. 2, serves formal verification and analysis. Formal verification is motivated by the many steps incorporating some randomized transformations in our scheme, ranging from encryption (with the coding scheme, gate selection during encoding, and the encryption key all being randomized), over construction of the EC and CC, to the system-level integration of the FC, all with their own intermediate key-bits. Still, all steps are correct-by-construction – verification is only employed as "sanity check" for our fully automated end-to-end flow. In fact, all LE runs pass verification; proofs are provided in our release [25].

3.6 Implementation

Coding: The encryption stage and all processes for compiling the different wrappers are implemented in *Python*. Synthesis processes are driven by *tcl* scripts, with synthesis itself conducted by Synopsys DC. Data management, including arrangement of the various circuit netlists across the different stages,

is implemented by *bash* scripts. Importantly, the whole method is fully automated.

Tooling: We utilize Cadence Conformal for verification, and Synopsys DC for synthesis and design analysis. Further details, in particular for the security analysis, are provided in Sec. 4.

Runtimes: Running the method end-to-end requires only tens of minutes across all benchmarks. Since all key steps for circuit design are underpinned by commercial-grade synthesis, runtimes scale graciously for larger benchmarks.

Release: We provide a full open-source release of our method at [25]. This includes all scripts/codes, as well as end-to-end examples for LE implementations on all benchmarks.[4]

3.7 Illustrative End-to-End Example of Logic Encryption

To demonstrate the full LE flow, we present a compact example using an original circuit (OC) with three inputs a, b, c and one primary output:

$$PO_{OC} = (a \wedge b) \vee c.$$

Stage Ⓐ : Construction of the Encrypted Circuit (EC)

A1) Coding Scheme. We use the NAND/NOR coding scheme:

$$\text{NAND} \mapsto 0, \qquad \text{NOR} \mapsto 1.$$

A2) Resynthesis and Encoding. The OC is rewritten using only NAND and NOR gates. One valid representation is:

$$t_1 = \text{NAND}(a, b)',$$
$$PO_{OC} = \text{NOR}(t_1, c)'.$$

Assume the resynthesized OC contains three gates in the memorized randomized order

$$[g_1 = \text{NAND},\ g_2 = \text{NAND},\ g_3 = \text{NOR}],$$

yielding the plaintext

$$P = 0\,0\,1.$$

[4]To support blind peer-review, or more precisely to protect the novelty of our work until official publication, not all details and files are included yet, but will be.

A3) AES Encryption of Encoded Gates. The plaintext is padded to 128 bits and encrypted with AES-128 using a random, not-stored key:

$$C = \mathrm{AES}_{K_{\mathrm{AES}}}(P_{\mathrm{padded}}).$$

For illustration, assume the first three ciphertext bits are:

$$C = 1\,0\,1.$$

A4) Decoding Ciphertext to Gates. Each ciphertext bit is decoded using the same codebook:

$$1 \rightarrow \mathrm{NOR}, \qquad 0 \rightarrow \mathrm{NAND}.$$

Thus the basic EC becomes:

$$[g_1^{EC} = \mathrm{NOR},\ g_2^{EC} = \mathrm{NAND},\ g_3^{EC} = \mathrm{NOR}].$$

A5) Resynthesis. The basic EC is resynthesized using the full gate library, producing the intermediate EC.

Stage Ⓑ : Construction of the Correction Circuit (CC)

For the single primary output, the CC logic is defined as:

$$PO_{CC} = PO_{OC} \oplus PO'_{EC},$$

where PO'_{EC} is either buffered or inverted. Let the randomized choice be inversion, giving:

$$PO'_{EC} = \overline{PO_{EC}}, \qquad K_{CC} = 1.$$

Synthesis yields the final CC.

Stage Ⓒ : Randomization of the EC

The intermediate EC's primary output is again randomly buffered or inverted. Assume buffering is selected:

$$PO_{EC,\mathrm{new}} = PO_{EC}, \qquad K_{EC} = 0.$$

Synthesis yields the final EC.

Stage Ⓓ : System-Level Integration

Each PO of the EC and CC passes through a MUX-based KG structure before entering the XOR used to produce the final circuit (FC) output:

$$PO_{FC} = PO_{EC}^{(MUX)} \oplus PO_{CC}^{(MUX)}.$$

Let the randomized MUX assignments generate:

$$K_{MUX}(PO_{EC}) = 1, \qquad K_{MUX}(PO_{CC}) = 0.$$

Final-Key Derivation. The final key-bits are computed as:

$$K_{FC}(PO_{EC}) = K_{MUX}(PO_{EC}) \oplus K_{EC} = 1 \oplus 0 = 1, \tag{5}$$

$$K_{FC}(PO_{CC}) = K_{MUX}(PO_{CC}) \oplus K_{CC} = 0 \oplus 1 = 1. \tag{6}$$

Thus the final key is:

$$K_{FC} = (1, 1).$$

Stage Ⓔ : Verification Formal equivalence checking confirms:

$$PO_{FC} = PO_{OC},$$

for the correct final key K_{FC}, completing the end-to-end example.

Table 1: Basic Properties for Benchmark Circuits

Name	# Gates	Power [μW]	Timing [ns]	Area [μm^2]	# Key-Bits (This Work)
c7552	726	711.89	3.19	829.65	214
c6288	709	1.20e+03	3.57	1,287.97	64
c5315	666	564.07	1.12	740.81	246
c3540	484	351.63	1.52	521.89	44
c2670	266	203.55	1.05	297.65	128
c1908	189	185.44	1.15	210.14	50
c1355	182	245.87	0.76	234.61	64
b22_C	6,361	6.74e+03	5.19	7,274.30	1,514
b21_C	4,274	4.57e+03	5.22	4,759.54	1,024
b20_C	4,205	4.51e+03	5.10	4,714.58	1,024
b17_C	10,696	5.46e+03	5.18	11,891.26	2,890
b15_C	3,295	1.82e+03	3.86	3,666.81	898
b14_C	2,061	1.90e+03	4.84	2,254.62	490

4 Experimental Investigation

4.1 Scope and General Settings

Benchmarks: All experiments utilize two well-known, fully combinational circuit datasets: ISCAS-85 [47] and ITC-99 [48]. Importantly, this selection enables comparisons with prior art, which most often only support these datasets. In general, LE is readily compatible with any combinational circuit. LE can also support sequential circuits, e.g., when applied between register stages.

All benchmarks are realized in *bench* and *Verilog* format. For the latter, gate-level netlists are synthesized using the well-known *Nangate 45nm* technology library at typical corner. Basic properties are listed in Tab. 1. Importantly, the numbers of key-bits are dictated by the workings of LE (Sec. 3.4).

Prior Art: We compare our novel LE scheme to selected SOTA schemes for LL: TRLL [26], gDMUX [27], and LUT-L [30]. We obtain TRLL from the authors [26], gDMUX from [49], and LUT-L from [50]. For LUT-L, we configure it for LUTs with 2 inputs. For this and all other schemes, we further utilize their default settings. For fair comparison, both in terms of security and design overheads, we utilize the same numbers of key-bits (Tab. 1) for all schemes.

Dataset Generation I, Randomized Runs: Given the inherent randomness for all LL/LE schemes, we conduct multiple runs for each scheme and each benchmark. For example, for LE, we conduct 10 independent runs for the encryption stage, and 4 end-to-end runs for each of those, resulting in 40 FC instances per benchmark. All datasets are used for design analysis and security analysis; details for the latter are described below.

Metrics I, Design Analysis: We report average PPA overheads across all the randomized runs for all schemes. Area is quantified as standard-cells area, power as total power, and performance/timing as critical-path delay.

4.2 Settings for Security Analysis

Threat Model: Following recent SOTA works, e.g., [14–17, 29], we consider the oracle-less threat model. That is, attackers have only access to the protected netlist, but not to an operational IC. Attackers can readily differentiate between regular gates and KG structures in the protected netlist.

Setups for Attacks: As indicated, we conduct a thorough security analysis covering major threat vectors for oracle-less attacks. More specifically, we cover prediction of KG structures by OMLA [15], prediction of KG interconnects by MuxLink [16], joint prediction of KG structures and interconnects by resynthesis with SCOPE [14], and RE by GNN-RE [29]; we utilize respective open-source release for OMLA [51], MuxLink [49], resynthesis with SCOPE [52], and GNN-RE [53].[5]

Matching the scope of the various LL/LE schemes against the scope of the various attacks, we apply OMLA for TRLL and LE, MuxLink for gDMUX and LE, and both GNN-RE and resynthesis with SCOPE for all schemes, respectively. We employ the respective default setups, with customizations as required as follows.

For GNN-RE, we extend the feature vector to cover all gate types in the full library. We accordingly revise the generation of gate/node labels for circuits. For MuxLink on LE, we assume the attacker can access the system-level components individually from within the FC (Fig. 3).[6] Accordingly, link prediction is readily accessible to the attacker, whereas in the real-world, these components would be entangled by the synthesis process for the FC, making such a clear distinction much more difficult. Thus, we conduct a conservative worst-case analysis here. For resynthesis with SCOPE on LE, in addition to regular runs on LE where we tackle/attack the FC as synthesized, we follow a similar approach, also inspired by the general divide-and-conquer strategy proposed in [43]. As indicated, such worst-case analysis is prudent as the resilience of our LE scheme depends on both the obfuscation of the system-level components (EC and CC) and their interconnects within the FC.

Dataset Generation II, Security Analysis: To enable dataset handling by the various attacks, we follow the respective default settings, with customizations as required as follows.

For GNN-RE, we obtain a baseline dataset, by resynthesizing each benchmark netlist using various recipes, generating different structural variants for each. Doing so is important to strive for balancing the different classes/circuits for the library-centric learning approach of GNN-RE. We follow a similar approach for generation of LL/LE datasets, i.e., for each scheme, we pick from their randomized runs in a balanced manner, and we also conduct resynthesis (with the same settings, for fairness). For OMLA, we realize a library-centric training with leave-one-out approach.

Metrics II, Security Analysis: For the ML-driven attacks OMLA, MuxLink, and GNN-RE, we report the attacks' success rates as average test accuracy. For GNN-RE, considering the multi-class prediction problem, we also report F1 micro/macro scores. For baseline/standalone SCOPE as well as resynthesis with SCOPE, we report the attacks' success rates in terms of accuracy (AC) and key-prediction accuracy (KPA).[7] For baseline SCOPE, the key-bit inferences are directly evaluated; for resynthesis with SCOPE, majority voting across all SCOPE runs on all resynthesized circuits is done for each key-bit guess. In both cases, two complementary sets of key-bit predictions are obtained. For reporting of both metrics, we first obtain the maxima across these two complementary sets, and then we average across all randomized runs.

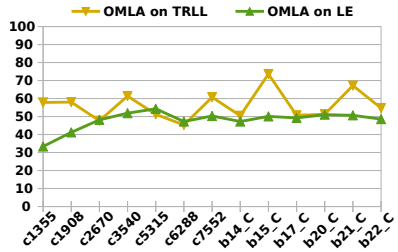

Figure 4: Test accuracy [%] for OMLA.

4.3 Security Analysis: KG Structures (OMLA)

The success rate for prediction of the correct KG structures, as in test accuracy for OMLA, is shown in Fig. 4.

LE demonstrates consistently strong resilience across all benchmarks, with success rates worse than random guessing (i.e., 50%) for smaller benchmarks and saturating well around random guessing for larger benchmarks. Compared to TRLL, the sole applicable contender for this attack setting, LE is superior by 1.17x on average. Furthermore, TRLL is sporadically challenged by larger benchmarks, indicating a lack of resilience scalability; the latter also aligns with more recent findings on TRLL [42].

In short, thanks to its encryption-driven and full-scale obfuscation, LE is more resilient against ML-based prediction of KG structures than prior SOTA (TRLL).

4.4 Security Analysis: KG Interconnects (MuxLink)

The success rate for prediction of the correct KG interconnects, as in test accuracy for MuxLink, is shown in Fig. 5.

LE exhibits good resilience, enforcing an accuracy often close to random guessing. However, for larger benchmarks, outliers reach to around 70%, i.e., some correlation exists for the system-level interconnects between EC and the CC. This is expected, given that (i) the EC and CC are orchestrated together to realize the FC and, more importantly, (ii) we employ a worst-case security analysis here where those system-level interconnects are readily accessible, whereas in the real-world, these are entangled/obfuscated by synthesis of the FC. Still, compared to gDMUX, the SOTA for MUX-based LL and sole applicable contender for this attack setting, LE is superior by 1.64x on average.

In short, thanks to its system-level use of MUX KG structures, LE is more resilient against ML-based prediction of KG interconnects than prior SOTA (gDMUX), even under worst-case settings.

[5] Other attacks like LIPSTICK [41] or NoBALL [40] are not released and, thus, not considered here.

[6] More specifically, both the EC and CC are separately available to the attacker. For each XOR at the final POs, we model all four possible combinations of interconnects underpinned by the two MUX KG structures related to $K_{MUX}(EC)$ and $K_{MUX}(CC)$. Moreover, we modify post-processing to consider such link combinations, with the combination having the highest joint probability presumed as the correct one. Finally, we explore different hop-sizes and find that $h = 4$ is most successful; we apply this setting for all MuxLink experiments for fair comparison.

[7] AC and KPA both quantify the correctly guessed key-bits over all key-bits; the difference is that KPA's denominator excludes unresolved key-bits, whereas AC's includes them. While AC is the key metric for the attacks' success, KPA provides further insights for the attacks' effectiveness: larger (lower) KPA values indicate more (less) confidence/precision for the resolved key-bits and, vice versa, more (less) clear decision boundaries against key-bits that cannot be resolved.

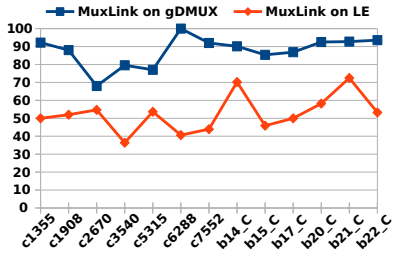

Figure 5: Test accuracy [%] for MuxLink.

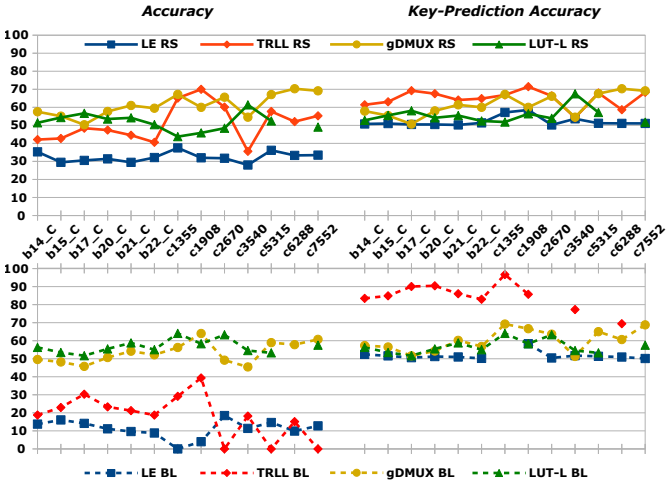

Figure 6: Accuracy and key-prediction accuracy [%] for resynthesis with SCOPE (RS; top) and baseline SCOPE (BL; bottom), respectively. For KPA, data points are missing for cases with AC=0%, where KPA is undefined. For LUT-L, for the benchmark c6288, data points are missing as the correct key was not returned by the related tool [50].

4.5 Security Analysis: Resynthesis with SCOPE

The success rates for joint prediction of KG structures and interconnects by baseline/standalone SCOPE and resynthesis with SCOPE, respectively, are shown in Fig. 6.

For the most relevant results, namely for AC for resynthesis with SCOPE, LE outperforms, on average, TRLL by 1.57x, gDMUX by 1.89x, and LUT-L by 1.60x, respectively. For AC for baseline SCOPE, LE even outperforms TRLL by 1.64x, gDMUX by 4.79x, and LUT-L by 5.10x, respectively, which implies that both gDMUX and LUT-L are already challenged by baseline SCOPE attacks, whereas LE (and LUT-L) induce more complex obfuscation, justifying further attack efforts via resynthesis with SCOPE. Importantly, LE is the only scheme that forces these further efforts well below the random-guessing threshold across all benchmarks, i.e., its resilience scales well for this sophisticated attack setting. For KPA for resynthesis with SCOPE vs. baseline SCOPE attacks, LE outperforms TRLL by 1.25x vs. 1.64x, gDMUX by 1.18x vs. 1.16x, and LUT-L by 1.07x vs. 1.10x, respectively, all on average. Again, LE is the only scheme that pushes KPA toward random-guessing domains across all benchmarks. This reconfirms the resilience of LE, as in resynthesis efforts are not contributing much toward more clear decision-making for SCOPE when resolving the key-bits.

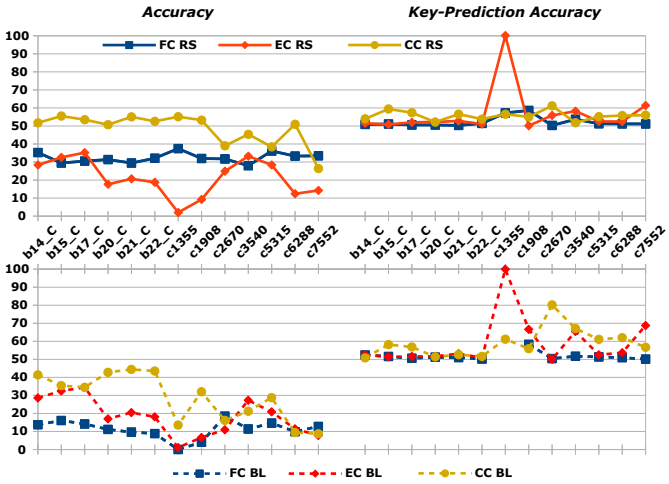

Figure 7: Accuracy and key-prediction accuracy [%] for attacking LE via resynthesis with SCOPE (RS; top) and baseline SCOPE (BL; bottom), respectively, all for the worst-case settings of attacking the EC and CC separately. Note that FC results are for the regular setting of attacking LE as is, carried over from Fig. 6. For KPA, data points are missing for cases with AC=0%, where KPA is undefined.

Upon further investigation, we note the following for LE. The construction of both EC and CC components, and their system-level integration with MUX KG structures, does not provide much leverage for synthesis-related attack efforts. This is because the integration of both the EC and CC is essentially based on whether POs/cones are used as-is vs. inverted at the system level, which is different from embedding KG structures throughout the netlist, as done by prior art in LL. Accordingly, for LE, the underlying circuit structures do not change significantly under SCOPE "hard-coding" the different possible combinations of key-bit values.

In Fig. 7, we provide further results for the outlined worst-case setting, i.e., without the system-level obfuscation realized by synthesizing the FC, where both EC and CC are attacked directly and separately for their respective intermediate key-bits, following the general divide-and-conquer strategy proposed in [43].

For the most relevant results, i.e., AC for resynthesis with SCOPE, attacking the EC achieves 21.44% AC (i.e., only 0.66x of the AC achieved when attacking the FC, or "AC for FC" for short), whereas attacking the CC achieves 48.35% AC (i.e., 1.49x AC for FC), all on average. For AC for baseline SCOPE, attacking the EC achieves 18.22% (i.e., 1.64x AC for FC), whereas attacking the CC achieves 28.58% (i.e., 2.57x AC for FC), respectively. These results are expected; there should be some benefit for attackers tackling the EC and CC directly and separately. Still, for KPA, values are close to random guessing, aside from the outlier for benchmark c1355, where the 100% KPA is a skewed result that arises from very few key-bits (i.e., 1 to 2) resolved at all, which shows from the corresponding AC value close to zero. As before, resynthesis efforts are not contributing much toward more clear decision-making for SCOPE.

In addition to these insights when compared to attacking the FC, these results also show that (i) the EC is more resilient than the CC, (ii) resynthesis efforts do not scale well for attacking the EC, and (iii),

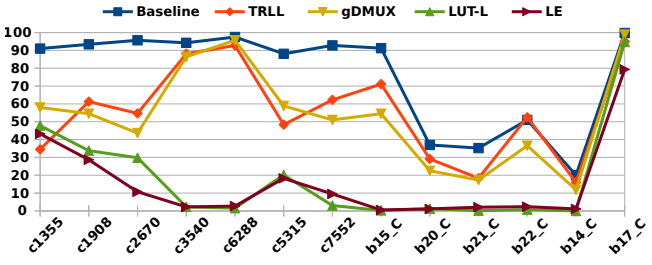

Figure 8: GNN-RE test accuracy [%].

Table 2: GNN-RE Multi-Class Prediction Metrics

	Baseline	TRLL [26]	gDMUX [27]	LUT-L [30]	LE [This]
F1 Micro	72.17%	60.16%	56.60%	28.81%	25.66%
F1 Macro	75.47%	55.85%	53.96%	14.40%	11.60%

most importantly, results remain in the random-guessing domain. Upon further investigation, we note the following. First, recall that results relate to the respective intermediate key-bits; thus, while the CC is more vulnerable when isolated, security analysis still requires a holistic view. That is, recovering the key for one component is insufficient; an attacker must successfully break both the EC and the CC. Second, circuit structures of the EC tend to be smaller and less complex than those of the CC, which can be explained by their respective construction steps: the EC is fully obfuscated by encryption, providing ample opportunities for synthesis to simplify the related circuitry, whereas the CC, while also derived from the EC, needs to realize restoration of the OC's functionality, which tends to become more complex. These aspects can explain the stronger resilience of the EC.

In short, thanks to the design and construction of both the EC and CC and their system-level integration with MUX KG structures, LE is more resilient against joint prediction of KG structures and interconnects than prior SOTA of LL. This holds true even for the worst-case setting where attackers can directly and separately tackle the EC and CC. This resilience is because the circuit structures of both EC and CC are operated either as-is vs. inverted at the system level, thereby remaining largely unaffected by synthesis-related restructuring efforts, which is different from the construction and operation of LL schemes.

4.6 Security Analysis: Reverse Engineering (GNN-RE)

The success rate for reverse engineering, as in test accuracy for GNN-RE, is shown in Fig. 8. We also report on the related multi-class prediction performance in Tab. 2. Note that baseline refers to the benchmarks as-is, i.e., without any LL/LE applied for obfuscation.

LE demonstrates a markedly stronger ability to obstruct GNN-RE, imposing the lowest F1 micro and macro scores, i.e., the lowest prediction performance, among all schemes (Tab. 2). Averaging the test accuracy across all benchmarks in Fig 8, LE outperforms TRLL by 3.58x, gDMUX by 3.41x, and LUT-L by 1.17x, respectively.

We also observe that LE (and LUT-L) exhibit strong resilience for both larger benchmarks with larger key-sizes and smaller ones. In contrast, TRLL and gDMUX face significant challenges when dealing with larger instances. The latter is because TRLL and gDMUX primarily introduce structural changes localized around the inserted KG structures, limiting their ability to obscure the entire design. Finally, note that b17_C is much more easily identified by GNN-RE across the board. This is due to the underlying functional similarities across ITC benchmarks [48]. LE still outperforms prior art for this challenging benchmark, namely by 1.21x on average.

In short, thanks to its encryption-driven and full-scale obfuscation, LE is more resilient against ML-based RE than prior SOTA.

4.7 Security Analysis: Discussion on Oracle-Guided Attacks

While conventional wisdom suggests oracle-guided attacks are stronger – due to the attacker's additional capabilities through oracle access – we may argue that the reverse holds when evaluating defenses. That is, some LL/LE scheme is powerful when readily resisting oracle-less attacks, whereas potential resilience against oracle-guided attacks only represents a weaker defense posture, as the latter assumes greater attacker capabilities. In other words, security assessment from the defender's viewpoint may rate attack models requiring less (vs more) resources or capabilities as more (vs less) powerful. As we propose a defense, we adopt this perspective on oracle-less attacks for our security analysis.

Nevertheless, in future work, we will investigate LE also in the oracle-guided threat model.[8] The fact that LE incorporates entanglement of system-level interconnects is potentially promising toward construction of SAT-hard problem instances, as outlined in [34]. Besides, LL/LE schemes can be extended via compound locking against oracle-guided attacks [1, 19], although doing so still requires some careful security analysis [5, 43, 54].

4.8 Design Analysis

PPA overheads are shown in Fig. 9.

Our proposed LE scheme significantly outperforms prior art for both area and power, and is a competitive runner-up against LUT-L for performance. More specifically, across the various benchmarks, area overheads over LE are 1.78x for TRLL, 2.41x for gDMUX, and 1.85x for LUT-L, respectively; power overheads are 2.06x for TRLL, 2.80x for gDMUX, and 1.87x for LUT-L, respectively; and performance overheads are 7.86x for TRLL, 71.56x for gDMUX, and -1.63x for LUT-L, respectively. When looking at the "bigger picture" of combined PPA overheads (i.e., overheads for area, power, and performance are summed up separately for each benchmark), LE outperforms, on average, TRLL by 2.45x, gDMUX by 8.72x, and LUT-L by 1.55x, respectively.

We observe that PPA overheads for LE, and to some degree also for LUT-L, often scale graciously for both larger circuits and key-sizes as well as smaller ones. In contrast, for power and area, TRLL is challenged by larger instances and gDMUX is sporadically challenged by both larger and smaller instances. For performance, TRLL is more challenged by smaller instances, whereas gDMUX shows

[8] Since doing so would require us to obtain, configure, run, and evaluate our experiments for various other LL schemes proposed against oracle-guided attacks, such efforts would be considerable and, again, considered as scope for future work.

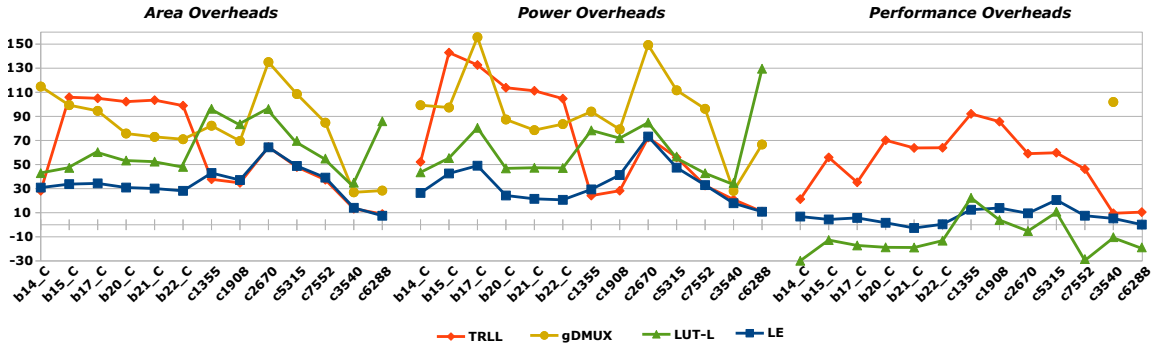

Figure 9: Average PPA overheads [%] for all considered schemes. Note that datapoints not shown (i.e., for performance for gDMUX) are exceeding the upper limit of the plot, i.e., 160%.

excessive overheads, namely 471.63% on average. The latter is because gDMUX selects data inputs for its MUX-based KG structures throughout the entire netlist. Thus, especially for larger key-bit sizes, most timing paths become iteratively entangled by gDMUX, which significantly lengthens the critical path. In contrast, critical paths are not impacted much for LE, as the main components (EC and CC) are orchestrated in parallel.

In short, thanks to its simple circuit design and synthesis-driven implementation – and despite its full-scale obfuscation – LE is competitive against SOTA in terms of PPA overheads.

5 Discussion

Roy *et al.*'s EPIC framework [55] was the first to incorporate cryptography into logic protection, but its use is restricted to the secure *delivery* of the activation key. EPIC embeds key-programmable gates into the design and uses asymmetric cryptography (e.g., RSA) only to encrypt the key that enables correct operation; once the key is loaded, the circuit structure is unchanged and fully visible, as no part of the logic itself is encrypted or structurally transformed. Later works apply cryptography-inspired mechanisms at finer granularity. *Encrypt Flip-Flop* [56] protects sequential logic by inserting MUXes at selected flip-flop outputs; key-bits select between true and complemented values so that incorrect keys propagate corrupt state values, yet the combinational logic structure remains entirely exposed. *Hardware Functional Obfuscation* [57] introduces FeFET-based active interconnects that can be programmed to behave as an inverter or buffer, allowing small logic regions to be reconfigured at run time. Although this provides a hardware-level means of concealing local behavior unless the correct configuration is programmed, only limited logic blocks are affected, and the original netlist is otherwise left intact. Across these works, cryptography or logic programmability is used to guard specific key-controlled elements, not to encrypt or transform the full logic representation.

In contrast, our LE methodology employs cryptography to protect the *entire* logic rather than isolated key-gates or reconfigurable segments. We encode the full gate-level description of the design, apply a block cipher (e.g., AES) to produce ciphertext that directly determines the encrypted circuit structure, and decode this ciphertext into a synthesized encrypted circuit whose internal gate types, connectivity, and functional behavior are all cryptographically randomized. This process obfuscates the circuit end-to-end: the logic is rearranged, structurally altered, and rendered unintelligible without the decryption machinery and correct key. Whereas prior works preserve the original netlist and merely alter its activation or local behavior, our approach remodels the netlist itself under cryptographic control, achieving comprehensive logic-level obfuscation capable of resisting stronger adversaries, including oracle-less reverse engineering, structural analysis, and resynthesis-based attacks.

6 Conclusions and Future Work

We have presented a first-of-its-kind work for actual LE. Recall that LL handles the integration of KG structures more as an afterthought, leaving them vulnerable to various attacks. In contrast, our approach to LE truly obfuscates the circuit's structure and functionality at full scale. With our extensive analysis, we demonstrate the promise of our LE scheme. Ours consistently outperforms prior art against a range of SOTA oracle-less attacks covering crucial threat vectors, all with lower PPA overheads. For example, LUT-L [30], the only contender to some degree, is still inferior to LE, namely by 1.17x for GNN-RE, by 1.60x/5.10x for resynthesis and/or SCOPE, and by 1.55x for PPA, respectively. The fact that LUT-L utilizes redaction and by doing so differs from traditional LL approaches reiterates the fallacies of prior art. LE excels over LUT-L thanks to its system-level obfuscation, whereas LUT-L is still limited to local redaction/obfuscation. Finally, we provide a full open-source release [25], fostering reproducibility and future work.

We envision multiple directions for future work with LE. First, we shall explore to also encode and encrypt the interconnects, together with the logic/gates. Second, we shall extend LE for the oracle-guided threat model. Third, we shall apply LE for larger and sequential circuits, including modern chiplet systems.[9]

References

[1] Abhishek Chakraborty, Nithyashankari Gummidipoondi Jayasankaran, Yuntao Liu, Jeyavijayan Rajendran, Ozgur Sinanoglu, Ankur Srivastava, Yang Xie, Muhammad Yasin, and Michael Zuzak. Keynote: A disquisition on logic locking. *Trans. Comp.-Aided Des. Integ. Circ. Sys.*, 39(10):1952–1972, 2019.

[9] Our LE method lends itself readily to the latter: the EC and the CC can be realized as two third-party chiplets, whereas secure system integration can be subsequently and separately realized by an active interposer [58]. Notably, such an implementation and supply-chain setup would leave the chiplet providers, which may be acting malicious, without access to the system-level oracle, thereby still hindering oracle-guided attacks.

[2] Sonia Akter, Kasem Khalil, and Magdy Bayoumi. A survey on hardware security: Current trends and challenges. *IEEE Access*, pages 77543–77565, 2023.

[3] Mark Tehranipoor, Nitin Pundir, Nidish Vashistha, and Farimah Farahmandi. *Hardware security primitives*. Springer, 2023.

[4] Mingfu Xue, Chongyan Gu, Weiqiang Liu, Shichao Yu, and Máire O'Neill. Ten years of hardware trojans: a survey from the attacker's perspective. *IET Computers & Digital Techniques*, 14(6):231–246, 2020.

[5] Benjamin Tan, Ramesh Karri, Nimisha Limaye, Abhrajit Sengupta, Ozgur Sinanoglu, Md Moshiur Rahman, Swarup Bhunia, Danielle Duvalsaint, R. D., Blanton, Amin Rezaei, Yuanqi Shen, Hai Zhou, Leon Li, Alex Orailoglu, Zhaokun Han, Austin Benedetti, Luciano Brignone, Muhammad Yasin, Jeyavijayan Rajendran, Michael Zuzak, Ankur Srivastava, Ujjwal Guin, Chandan Karfa, Kanad Basu, Vivek V. Menon, Matthew French, Peilin Song, Franco Stellari, Gi-Joon Nam, Peter Gadfort, Alric Althoff, Joseph Tostenrude, Saverio Fazzari, Eric Breckenfeld, and Kenneth Plaks. Benchmarking at the frontier of hardware security: Lessons from logic locking, 2020.

[6] Jarrod A. Roy, Farinaz Koushanfar, and Igor L. Markov. Ending piracy of integrated circuits. *Computer*, 43(10):30–38, 2010.

[7] Hai Zhou, Amin Rezaei, and Yuanqi Shen. Resolving the trilemma in logic encryption. In *Proc. Int. Conf. Comp.-Aided Des.*, pages 1–8, 2019.

[8] Jugal Gandhi, Diksha Shekhawat, M Santosh, and Jai Gopal Pandey. Logic locking for ip security: A comprehensive analysis on challenges, techniques, and trends. *Computers & Security*, 129:103196, 2023.

[9] Deepak Sirone and Pramod Subramanyan. Functional analysis attacks on logic locking. *Trans. Inf. Forens. Sec.*, 15:2514–2527, 2020.

[10] Bulbul Ahmed, Sazadur Rahman, Kimia Zamiri Azar, Farimah Farahmandi, Fahim Rahman, and Mark Tehranipoor. Seemless: Security evaluation of logic locking using machine learning oriented estimation. In *Proc. Great Lakes Symp. VLSI*, pages 489–494, 2024.

[11] Anand Raj, Nikhitha Avula, Pabitra Das, Dominik Sisejkovic, Farhad Merchant, and Amit Acharyya. Deepattack: A deep learning based oracle-less attack on logic locking. In *Proc. Int. Symp. Circ. Sys.*, pages 1–5, 2023.

[12] Levent Aksoy, Muhammad Yasin, and Samuel Pagliarini. Kratt: Qbf-assisted removal and structural analysis attack against logic locking. In *Proc. Des. Autom. Test Europe*, pages 1–6, 2024.

[13] Abdul Khader Thalakkattu Moosa, Benjamin Tan, and Ramesh Karri. Scaling attacks on large logic-locked designs. *Trans. Comp.-Aided Des. Integ. Circ. Sys.*, 2023.

[14] Felipe Almeida, Levent Aksoy, Quang-Linh Nguyen, Sophie Dupuis, Marie-Lise Flottes, and Samuel Pagliarini. Resynthesis-based attacks against logic locking. In *Proc. Int. Symp. Qual. Elec. Des.*, pages 1–8, 2023.

[15] Lilas Alrahis, Satwik Patnaik, Muhammad Shafique, and Ozgur Sinanoglu. OMLA: An oracle-less machine learning-based attack on logic locking. *Trans. Circ. Sys. II*, 69(3):1602–1606, 2022.

[16] Lilas Alrahis, Satwik Patnaik, Muhammad Shafique, and Ozgur Sinanoglu. MuxLink: Circumventing learning-resilient mux-locking using graph neural network-based link prediction. In *Proc. Des. Autom. Test Europe*, pages 694–699, 2022.

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

[18] Prabuddha Chakraborty, Jonathan Cruz, and Swarup Bhunia. Sail: Machine learning guided structural analysis attack on hardware obfuscation. In *Proc. Asian Hardw.-Orient. Sec. Trust Symp.*, pages 56–61, 2018.

[19] Mark Tehranipoor, Kimia Zamiri Azar, Navid Asadizanjani, Fahim Rahman, Hadi Mardani Kamali, and Farimah Farahmandi. Advances in logic locking. In *Hardware Security: A Look into the Future*, pages 53–142. Springer, 2024.

[20] Milan Vojvoda, Viliam Hromada, Filip Janikovskỳ, Jozef Kučerák, and Matúš Jókay. Experimental evaluation of sat attack on logic locking. 2024.

[21] Zeng Wang, Lilas Alrahis, Animesh Basak Chowdhury, Dominik Germek, Ramesh Karri, and Ozgur Sinanoglu. Optilock: Automated optimization of learning-resilient logic locking. *IEEE Access*, 2025.

[22] Zeng Wang, Lilas Alrahis, Dominik Sisejkovic, and Ozgur Sinanoglu. Autolock: Automatic design of logic locking with evolutionary computation. In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, pages 200–202. IEEE, 2023.

[23] Jeyavijayan Rajendran, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. Security analysis of logic obfuscation. In *Proc. Des. Autom. Conf.*, pages 83–89, 2012.

[24] Lilas Alrahis, Satwik Patnaik, Faiq Khalid, Muhammad Abdullah Hanif, Hani Saleh, Muhammad Shafique, and Ozgur Sinanoglu. Gnnunlock: Graph neural networks-based oracle-less unlocking scheme for provably secure logic locking. In *Proc. Des. Autom. Test Europe*, pages 780–785, 2021.

[25] Anonymous. Logic encryption: This time for real. https://anonymous.4open.science/r/LogicEncryption-TimeForReal-E1C1/README.md.

[26] Nimisha Limaye, Emmanouil Kalligeros, Nikolaos Karousos, Irene G. Karybali, and Ozgur Sinanoglu. Thwarting all logic locking attacks: Dishonest oracle with truly random logic locking. *Trans. Comp.-Aided Des. Integ. Circ. Sys.*, 40(9):1740–1753, 2021.

[27] Dominik Sisejkovic, Farhad Merchant, Lennart M. Reimann, and Rainer Leupers. Deceptive logic locking for hardware integrity protection against machine learning attacks. *Trans. Comp.-Aided Des. Integ. Circ. Sys.*, 41(6):1716–1729, 2022.

[28] Kaveh Shamsi and Rajesh Kumar Datta. TIPLock: Key-compressed logic locking using through-input-programmable lookup-tables. In *Proc. Des. Autom. Test Europe*, pages 1–2, 2023.

[29] Lilas Alrahis, Abhrajit Sengupta, Johann Knechtel, Satwik Patnaik, Hani Saleh, Baker Mohammad, Mahmoud Al-Qutayri, and Ozgur Sinanoglu. GNN-RE: Graph neural networks for reverse engineering of gate-level netlists. *Trans. Comp.-Aided Des. Integ. Circ. Sys.*, 41(8):2435–2448, 2022.

[30] Alex Baumgarten, Akhilesh Tyagi, and Joseph Zambreno. Preventing ic piracy using reconfigurable logic barriers. *Des. Test*, 27(1):66–75, 2010.

[31] Nikhil Saxena. *Efficient Techniques for Logic Locking*. PhD thesis, University of Cincinnati, 2024.

[32] Sophie Dupuis, Nassim Riadi, Clémy Moroukian, Florence Azaïs, and Marie-Lise Flottes. Logic locking: Exploration of a new key-gate based on tristate logic. In *Proc. Lat.-Am. Test. Symp.*, pages 1–6, 2024.

[33] Fangzhou Wang, Qijing Wang, Bangqi Fu, Shui Jiang, Xiaopeng Zhang, Lilas Alrahis, Ozgur Sinanoglu, Johann Knechtel, Tsung-Yi Ho, and Evangeline F.Y. Young. Security closure of ic layouts against hardware trojans. In *Proc. Int. Symp. Phys. Des.*, 2023.

[34] Hadi Mardani Kamali, Kimia Zamiri Azar, Houman Homayoun, and Avesta Sasan. Interlock: an intercorrelated logic and routing locking. In *Proc. ICCAD*, 2020.

[35] Jeyavijayan Rajendran, Huan Zhang, Chi Zhang, Garrett S Rose, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. Fault analysis-based logic encryption. *Trans. Comp.*, 64(2):410–424, 2013.

[36] Amin Rezaei, Yuanqi Shen, Shuyu Kong, Jie Gu, and Hai Zhou. Cyclic locking and memristor-based obfuscation against cycsat and inside foundry attacks. In *Proc. Des. Autom. Test Europe*, pages 85–90, 2018.

[37] Ziad El Sayed, Zeng Wang, Hana Selmani, Johann Knechtel, Ozgur Sinanoglu, and Lilas Alrahis. Graph neural networks for integrated circuit design, reliability, and security: Survey and tool. *ACM Computing Surveys*, 58(4):1–44, 2025.

[38] Fangfei Yang, Ming Tang, and Ozgur Sinanoglu. Stripped functionality logic locking with hamming distance-based restore unit (sfll-hd)–unlocked. *Trans. Inf. Forens. Sec.*, 14(10):2778–2786, 2019.

[39] Pramod Subramanyan, Sayak Ray, and Sharad Malik. Evaluating the security of logic encryption algorithms. In *Proc. Int. Symp. Hardw.-Orient. Sec. Trust*, pages 137–143, 2015.

[40] Praveen Karmakar, Anmoldeep Singh, Kartik Sharma, Chandan Karfa, and Sukanta Bhattacharjee. Noball: A novel bdd-based attack against logic locking. In *Proc. Int. Test Conf.*, pages 1–6, 2024.

[41] Yeganeh Aghamohammadi and Amin Rezaei. Lipstick: Corruptibility-aware and explainable graph neural network-based oracle-less attack on logic locking. In *Proc. Asia South Pac. Des. Autom. Conf.*, pages 606–611, 2024.

[42] Lakshmi Likhitha Mankali, Ozgur Sinanoglu, and Satwik Patnaik. INSIGHT: Attacking Industry-Adopted learning resilient logic locking techniques using explainable graph neural network. In *Proc. USENIX Sec. Symp.*, 2024.

[43] Felipe Almeida, Levent Aksoy, and Samuel Pagliarini. Resaa: A removal and structural analysis attack against compound logic locking. *Trans. VLSI Syst.*, 33(5):1348–1360, 2025.

[44] Akashdeep Saha, Sayandeep Saha, Siddhartha Chowdhury, Debdeep Mukhopadhyay, and Bhargab B Bhattacharya. Lopher: Sat-hardened logic embedding on block ciphers. In *Proc. Des. Autom. Conf.*, pages 1–6, 2020.

[45] Prithwish Basu Roy, Johann Knechtel, Akashdeep Saha, Saideep Sreekumar, Likhitha Mankali, Mohammed Nabeel, Debdeep Mukhopadhyay, Ramesh Karri, and Ozgur Sinanoglu. NiLoPher: Breaking a modern SAT-hardened logic-locking scheme via power analysis attack. Cryptology ePrint, Paper 2024/309, 2024.

[46] Tim Bücher, Lilas Alrahis, Guilherme Paim, Sergio Bampi, Ozgur Sinanoglu, and Hussam Amrouch. Appgnn: Approximation-aware functional reverse engineering using graph neural networks. In *Proc. Int. Conf. Comp.-Aided Des.*, 2022.

[47] Mark C Hansen, Hakan Yalcin, and John P Hayes. Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering. *Des. Test*, 16(3):72–80, 1999.

[48] Scott Davidson. ITC'99 benchmark circuits – preliminary results. In *Proc. Int. Test Conf.*, pages 1125–1125, 1999.

[49] Lilas Alrahis. MuxLink. https://github.com/lilasrahis/MuxLink.

[50] Kaveh Shamsi. Netlist encryption and obfuscation suite. https://bitbucket.org/kavehshm/neos.

[51] Lilas Alrahis. OMLA. https://github.com/DfX-NYUAD/OMLA.

[52] Felipe Almeida, Samuel Pagliarini, and Johann Knechtel. Resynthesis tool. https://github.com/DfX-NYUAD/resynthesis_attack.

[53] Lilas Alrahis. GNN-RE. https://github.com/DfX-NYUAD/GNN-RE.

[54] Nimisha Limaye, Satwik Patnaik, and Ozgur Sinanoglu. Fa-sat: Fault-aided sat-based attack on compound logic locking techniques. In *Proc. Des. Autom. Test Europe*, pages 1166–1171, 2021.

[55] Jarrod A Roy, Farinaz Koushanfar, and Igor L Markov. Epic: Ending piracy of integrated circuits. In *Proceedings of the conference on Design, automation and test in Europe*, pages 1069–1074, 2008.

[56] Rajit Karmakar, Santanu Chatopadhyay, and Rohit Kapur. Encrypt flip-flop: A novel logic encryption technique for sequential circuits. *arXiv preprint arXiv:1801.04961*, 2018.

[57] Tongguang Yu, Yixin Xu, Shan Deng, Zijian Zhao, Nicolas Jao, You Sung Kim, Stefan Duenkel, Sven Beyer, Kai Ni, Sumitha George, et al. Hardware functional obfuscation with ferroelectric active interconnects. *Nature communications*, 13(1):2235, 2022.

[58] Heechun Park, Jinwoo Kim, Venkata Chaitanya Krishna Chekuri, Majid Ahadi Dolatsara, Mohammed Nabeel, Alabi Bojesomo, Satwik Patnaik, Ozgur Sinanoglu, Madhavan Swaminathan, Saibal Mukhopadhyay, Johann Knechtel, and Sung Kyu Lim. Design flow for active interposer-based 2.5-d ics and study of risc-v architecture with secure noc. *Trans. Compon., Pack., Manuf. Tech.*, 10(12):2047–2060, 2020.