Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Pre-Fabrication to Post-Deployment: A Unified ML-Driven Pipeline for IC Logic Locking

Nahush Tambe

Ntambe@charlotte.edu

University of North Carolina at Charlotte

Fareena Saqib

University of North Carolina at Charlotte

Research Article

Keywords: Logic Locking, Machine Learning, STREAMLOCK, Stream Cipher, Dynamic Partial Reconfiguration (DPR), Hardware Security

Posted Date: September 7th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10766700/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Pre-Fabrication to Post-Deployment: A Unified ML-Driven Pipeline for IC Logic Locking

Nahush Tambe[1*] and Fareena Saqib[1*]

[1*]Dept. of Electrical and Computer Engineering, UNC Charlotte, Charlotte, NC, USA.

*Corresponding author(s). E-mail(s): Ntambe@charlotte.edu; Fsaqib@charlotte.edu;

Abstract

Every integrated circuit (IC) in the globally distributed semiconductor supply chain is a potential target for intellectual property (IP) piracy, reverse engineering, and hardware tampering. Logic locking is a widely adopted defense primitive, yet its effectiveness is constrained by three limitations: technique selection is largely manual and circuit-agnostic; static key-gate schemes are vulnerable to oracle-guided and structural attacks; and post-deployment security remains largely unaddressed. This paper presents a unified, four-stage machine learning (ML)-driven pipeline that addresses these limitations within a single automated framework spanning the complete IC protection lifecycle.

A supervised Random Forest classifier extracts structural features from gate-level netlists and predicts the most appropriate technique from a portfolio of five classical locking schemes, achieving up to 97% prediction accuracy. Circuits are then secured either through fault-guided key-gate insertion or through STREAMLOCK, a lightweight stream cipher-based mechanism whose continuously evolving keystream replaces the fixed key-to-output relationship exploited by oracle-guided and structural attacks. Dynamic Partial Reconfiguration (DPR) then enables runtime key-gate updates on FPGA fabric without system interruption.

Experimental evaluation across ISCAS'85 benchmarks confirms 100% functional correctness under the correct key. A three-way comparison among classical, STREAMLOCK, and Advanced Encryption Standard (AES)-based locking demonstrates that STREAMLOCK achieves up to 9.3× lower Look-Up Table (LUT) utilization and up to 20.8× lower logic power than AES, while delivering cryptographically stronger protection than classical schemes. DPR integration introduces only a 5.92% increase in total power consumption and a 0.011%

increase in LUT utilization. Together, these results establish a scalable and hardware-efficient IC protection pipeline.

Keywords: Logic Locking, Machine Learning, STREAMLOCK, Stream Cipher, Dynamic Partial Reconfiguration (DPR), Hardware Security

1 Introduction

The modern integrated circuit (IC) design and manufacturing ecosystem is globally distributed, with design, verification, and assembly operations routinely performed by geographically disparate entities. While this horizontal business model has driven unprecedented reductions in production cost and time-to-market, it has introduced a broad and growing attack surface. Untrusted foundries, assembly houses, and third-party IP (3PIP) vendors each represent potential entry points for adversarial exploitation [1, 2]. The consequences of such exploitation are severe: IP piracy results in lost revenue and competitive advantage; hardware Trojan insertion compromises functional integrity; and reverse engineering enables unauthorized cloning and overproduction. The semiconductor industry's losses to IP theft are estimated between $225 billion and $600 billion annually, underscoring the urgency of efficient hardware protection mechanisms [3].

Among the countermeasures developed to address supply chain vulnerabilities, logic locking has risen to prominence as one of the most extensively studied hardware security primitives, owing to its ability to enforce protection at the pre-silicon stage [4, 5]. By embedding key-controlled gates, such as XOR/XNOR elements or multiplexers, directly into the circuit netlist prior to fabrication, logic locking binds correct circuit behavior to a secret key held exclusively by the IP owner. Any entity that gains access to the locked netlist without the authorized key, whether through a compromised fabrication facility or an untrusted supply chain intermediary, obtains a design that is structurally intact but functionally incorrect [6].

Despite a decade of active research producing a rich portfolio of locking techniques, their practical adoption in industry remains limited [7]. What constrains broader deployment, therefore, is not a deficit of viable locking strategies, but the continued reliance on manual, expertise-dependent decision-making that neither scales with circuit complexity nor guarantees consistent security outcomes. As circuits grow more heterogeneous, mixing combinational, sequential, FSM-based, and arithmetic logic within a single design, the consequences of an ill-suited technique selection extend beyond inefficiency to outright security compromise.

Equally concerning is the security of the locking mechanisms themselves. Traditional static key-gate techniques, including XOR/XNOR-based locking and SAT-resilient variants, embed a fixed, time-invariant relationship between the secret key and circuit outputs, a structural property that is systematically exploitable by Boolean Satisfiability (SAT)-based attacks [8]. While techniques such as SARLock and Anti-SAT [9, 10] increased the computational complexity of SAT-based key recovery, they

were subsequently shown to be vulnerable to removal and bypass attacks that circumvent the locking mechanism entirely without recovering the key [11]. More broadly, the static nature of all classical key-gate schemes exposes them to structural analysis, machine learning (ML)-based attacks, and function approximation techniques that exploit the fixed Boolean relationship between inputs, keys, and outputs [12]. These developments motivate the exploration of cryptographic primitives, specifically lightweight constructs that introduce dynamic, time-varying key-dependent behavior, as a more fundamentally secure alternative to static gate insertion.

Yet even a cryptographically effective locking mechanism offers no protection once a design leaves the fabrication stage, as the post-deployment security of locked designs remains largely unaddressed. Once a locked netlist is fabricated and deployed on Field Programmable Gate Array (FPGA) or Application-Specific Integrated Circuit (ASIC) fabric, the static configuration of its key-gate regions persists unchanged for the operational lifetime of the device. This provides a fixed and increasingly well-characterized attack surface for side-channel analysis, bitstream probing, and long-term statistical cryptanalysis. FPGA bitstream encryption provides a first layer of protection but has been demonstrated to be insufficient against targeted physical attacks and deliberate bitstream modifications [13]. This operational reality demands a mechanism by which the effective locking configuration of a deployed design can be updated post-fabrication, without system downtime, resynthesis, or re-deployment, in response to emerging attack intelligence or periodic key rotation requirements. The absence of such a mechanism in existing logic locking frameworks represents a fundamental gap between laboratory security evaluations and the demands of real-world, field-deployed hardware systems.

Rather than treating these as separate problems, this paper addresses them collectively within a single, automated pipeline spanning the complete IC protection lifecycle from pre-fabrication technique selection to post-deployment runtime security. A supervised Random Forest classifier automatically selects the most appropriate locking technique from a portfolio of five classical schemes based on structural features extracted from each circuit's gate-level netlist. The selected technique is implemented through one of two parallel paths. For classical protection, fault-guided key-gate insertion is performed by targeting nodes of maximum fault propagation impact. For circuits requiring stronger cryptographic guarantees, STREAMLOCK [14], a novel Nonlinear Feedback Shift Register (NLFSR)-based stream cipher-driven locking mechanism, is deployed instead, introducing dynamic, pseudorandom keystream masking into the circuit's combinational logic. The locked design is then deployed on FPGA fabric with Dynamic Partial Reconfiguration (DPR) enabled, permitting key-gate regions to be silently updated at runtime.

Beyond the pipeline itself, this paper presents the first systematic three-way area comparison among classical, STREAMLOCK-based, and AES-based locking approaches across benchmark circuits of varying complexity, supplemented by a detailed power consumption analysis between STREAMLOCK and AES. This comparative analysis provides the hardware security community with empirically grounded guidance on the area and power trade-offs of each approach as circuit complexity scales. The results establish that STREAMLOCK occupies a practically significant

middle ground by delivering cryptographically stronger protection than static classical schemes while incurring substantially lower area and power overhead than AES-based alternatives.
The key contributions of this paper are summarized as follows:

- **Unified Automated Pipeline**: An end-to-end, ML-driven framework that integrates architecture-aware technique selection, classical and cryptographic key-gate implementation, and runtime reconfigurable post-deployment security into a single automated IC protection pipeline.
- **Architecture-Aware ML-Based Technique Selection**: A supervised Random Forest classifier that automatically predicts the most appropriate locking technique across five classical schemes for diverse circuit architectures.
- **STREAMLOCK**: A lightweight stream cipher-driven locking mechanism built on a cross-coupled NLFSR network and nonlinear output combiner, introducing dynamic, time-varying key-dependent masking that resists conventional SAT-based key recovery.
- **DPR-Enhanced Post-Deployment Security**: Runtime key-gate reconfiguration via Dynamic Partial Reconfiguration on FPGA fabric, providing a continuously adaptive post-deployment defense layer with minimal area and power overhead.
- **Three-Way Area-Overhead Analysis**: The first empirical area and power comparison of classical, stream cipher-based, and AES-based locking across benchmark circuits of varying complexity, establishing practical area-overhead trade-offs to guide protection strategy selection.

The remainder of this paper is organized as follows. Section 2 establishes the technical foundations of the proposed pipeline. Section 3 reviews related work in logic locking, cryptographic hardware security, and FPGA-based runtime protection. Section 4 presents the unified pipeline architecture and its four constituent stages. Section 5 presents and analyzes the experimental results, including a three-way comparative analysis among classical, stream cipher-based, and AES-based locking. Section 6 concludes the paper and outlines directions for future work.

2 Preliminaries

This section establishes the technical foundations underlying the unified pipeline presented in this work. As hardware security threats evolve from simple reverse engineering to sophisticated oracle-guided cryptanalysis, effective IP protection demands a multi-layered, architecture-sensitive defense strategy that spans the full silicon lifecycle. The subsections that follow establish the fundamentals of classical logic locking, the architectural principles of runtime reconfigurable security, and the algorithmic foundations of lightweight cryptographic primitives. These form the three technical pillars upon which the pipeline is constructed.

2.1 Logic Locking Fundamentals

Logic locking is a pre-fabrication hardware security primitive that obscures a circuit's intended functionality by inserting key-dependent gates into its gate-level netlist, such

that correct behavior is contingent upon the application of an authorized secret key [4]. Formally, a logic-locked circuit C_L is derived from an original circuit C by augmenting it with n key-input bits $K = \{k_1, k_2, \ldots, k_n\}$, such that:

$$C_L(X, K) = \begin{cases} C(X) & \text{if } K = K^* \\ C'(X) \neq C(X) & \text{if } K \neq K^* \end{cases} \tag{1}$$

where X denotes the primary input vector, K^* is the correct secret key, and $C'(X)$ represents the corrupted output function produced under any unauthorized key. The security of a locked circuit is evaluated along two primary axes: output corruptibility, quantified by the Hamming distance between correct and incorrect outputs across all input patterns, and attack resilience, assessed against Boolean Satisfiability (SAT)-based key-recovery attacks [8].

The canonical threat model assumes an adversary with complete structural knowledge of the locked netlist and query access to a functional oracle capable of producing correct input-output pairs. The diversity of attack vectors that have emerged under this model, including SAT-based [8], removal-based [11], and structural analysis attacks [12], motivates the technique-specific, architecture-aware selection strategy central to this work. The following subsections characterize the locking techniques considered within the automated selection framework.

2.1.1 Random Logic Locking (RLL)

Random Logic Locking (RLL) constitutes the earliest formalized approach to logic obfuscation, operating through the indiscriminate insertion of XOR/XNOR gates at non-deterministic nodes in the post-synthesis netlist [15]. Each inserted gate consumes one bit of the secret key, inverting or preserving the traversing signal depending on whether the corresponding key bit is logic-1 or logic-0. While early evaluations on the ISCAS'85 combinational benchmark suite demonstrated that even modest insertion densities could effectively prevent the occurrence of multiple identical correct keys [15], RLL's gate placement is architecturally agnostic. It disregards logic cone depth, signal reconvergence, and fan-out characteristics; as a result, its security guarantees deteriorate rapidly under SAT-based analysis [8].

2.1.2 SAT-Attack Resilient Logic Locking (SARLock)

SAT-Attack Resilient Logic Locking (SARLock) addresses the principal vulnerability of RLL by structurally constraining the number of input patterns for which an incorrect key produces observable output corruption [9]. The mechanism augments the locked circuit with a comparator network that generates a flip signal exclusively for one specific input-key combination, ensuring that any erroneous key corrupts the circuit output on at most one primary input pattern. This construction forces a SAT solver to expend one full iteration per eliminated incorrect key candidate, substantially increasing attack complexity relative to classical XOR-based schemes. SARLock is particularly well-suited to lightweight circuit designs where SAT-solver resilience is the primary security objective and high output corruption is not a strict requirement [16, 17].

2.1.3 Fault Analysis Logic Locking (FALL)

Fault Analysis Logic Locking (FALL) draws upon classical stuck-at fault propagation theory from IC testing to guide key-gate placement towards nodes of maximum fault impact [18]. Rather than inserting gates at arbitrary locations, FALL employs fault simulation to rank circuit nodes by their fault propagation potential, with key-gates hierarchically inserted at the highest-ranked nodes. This targeted placement maximizes the spread of erroneous signal values through the logic cone under any incorrect key, achieving a Hamming distance of approximately 50% between correct and incorrect output vectors. FALL is particularly well-suited to purely combinational circuits where fault propagation paths are well-defined and uninterrupted by sequential state elements [16–18].

2.1.4 Stripped Functionality Logic Locking (SFLL)

Stripped Functionality Logic Locking (SFLL) departs from gate-insertion strategies by excising a carefully selected functional sub-component from the circuit and replacing it with a key-activated restore unit [19]. The restore unit validates the applied key and reinstates the stripped functionality only upon correct authentication; all other key values produce corrupted circuit behavior. SFLL offers provable resilience against SAT-based attacks for the stripped functional component, making it theoretically stronger than insertion-based approaches for certain circuit types.

2.1.5 Timing Aware Logic Locking (TALL)

Timing Aware Logic Locking (TALL) addresses the temporal characteristics of a circuit, a dimension overlooked by conventional locking techniques. Existing approaches operate exclusively in the logical domain, leaving critical path delays and timing margins as an exploitable side channel. TALL introduces key-dependent timing-ambiguous (TA) cells into the design's critical and near-critical timing paths [16, 17, 20]. Under the correct key, TA cells preserve the original timing profile of the circuit; under any incorrect key, delay paths are perturbed in a key-dependent manner, simultaneously corrupting both the logical outputs and the temporal behavior of the design. This dual-domain obfuscation makes TALL particularly effective for sequential circuits with stringent timing constraints.

2.1.6 Cyclic Logic Locking (CLL)

Cyclic Logic Locking (CLL) increases structural complexity by deliberately introducing feedback paths into an otherwise acyclic circuit netlist [21]. When the correct key is applied, inserted feedback paths are neutralized, preserving the intended combinational or sequential behavior of the design. Under an incorrect key, the active feedback loops induce functional inconsistencies, oscillatory behavior, or state-space divergence that renders the circuit inoperable. CLL is most naturally applicable to sequential and FSM-based circuits, where existing architectural feedback paths provide a structural substrate into which additional locking cycles can be integrated without disrupting verified functionality [16, 17].

2.1.7 Corrupt and Correct Logic Locking (CCLL)

Corrupt and Correct Logic Locking (CCLL) combines distributed output corruption with deterministic functional restoration [22]. Additional key-controlled logic elements, including XOR, AND, and OR gates, are distributed throughout the netlist, collectively corrupting circuit outputs across a broad range of input patterns when an incorrect key is applied. A complementary correction network, activated exclusively by the correct key, restores intended functionality without exhibiting structural regularity that could be exploited by pattern-matching or structural analysis tools. CCLL is well-suited to hybrid circuits that combine combinational and sequential logic, where broad corruption coverage and deterministic correctability are simultaneously required [16, 17].

2.2 Dynamic Partial Reconfiguration

Dynamic Partial Reconfiguration (DPR) is a capability of modern FPGA architectures that permits selective modification of designated logic regions, referred to as Reconfigurable Partitions (RPs), at runtime, while the remainder of the implemented design continues operating without interruption [23, 24]. The hardware design is partitioned into a static region, which houses the primary system logic and remains fixed throughout operation, and one or more RPs that can be independently updated by loading partial bitstreams at runtime. In the context of logic locking, this property enables key-gate regions to be reconfigured on-the-fly, for example, toggling between XOR and XNOR gate configurations, without resynthesis, re-implementation, or system downtime.

The security benefit of DPR is twofold. First, the temporal variability introduced by runtime reconfiguration eliminates the static attack surface that persists in conventionally deployed locked designs, frustrating side-channel analysis and long-term statistical attacks that depend on a fixed hardware configuration. Second, DPR enables key-gate updates to be issued post-deployment, a capability directly relevant to field-deployed systems in defense, aerospace, and critical infrastructure where periodic key rotation without system shutdown is operationally essential.

2.3 Lightweight Cryptographic Primitives

Lightweight cryptographic primitives have emerged as a practical alternative to block cipher-based hardware protection, offering strong security guarantees with significantly reduced area and power requirements [25]. Stream ciphers, in particular, generate pseudorandom keystreams by evolving a compact internal state through a nonlinear feedback function, producing bit sequences that are XORed with circuit signals to introduce key-dependent masking. Unlike block ciphers such as AES, which process fixed-length data blocks through computationally intensive substitution-permutation networks, stream ciphers operate sequentially and require minimal logic resources, making them well-suited to resource-constrained environments [26].

Nonlinear Feedback Shift Registers (NLFSRs) form the computational core of most hardware-oriented stream ciphers. An NLFSR of length L maintains an internal state

$S = (s_0, s_1, \ldots, s_{L-1}) \in \mathbb{F}_2^L$ that evolves according to a nonlinear Boolean feedback function f:

$$s_{t+L} = f(s_t, s_{t+1}, \ldots, s_{t+L-1}) \tag{2}$$

The nonlinearity of f is critical for cryptographic security, preventing internal state recovery through linear algebraic methods. Well-studied NLFSR-based stream ciphers, including Trivium [27], Grain [28], and MICKEY [29], have been formally analyzed for resistance against linear, differential, and correlation attacks. Their inherent sensitivity to key and initialization vector (IV) perturbations is of particular relevance to logic locking: a single-bit deviation in either the key or IV propagates rapidly through the NLFSR feedback network, producing a keystream that diverges unpredictably from the authorized sequence and causing widespread output corruption across the protected circuit.

3 Literature Review

The evolution of logic locking as a hardware security primitive reflects an ongoing arms race between increasingly sophisticated protection mechanisms and adversarial techniques developed to circumvent them. This section surveys the existing body of work across four dimensions: the progression of classical locking techniques and their known vulnerabilities, the emergence of cryptographic primitives as locking mechanisms, the application of machine learning to automate hardware protection decisions, and the persistent automation gap that motivates the unified pipeline proposed in this work.

3.1 Evolution of Logic Locking Primitives

The earliest logic locking implementations relied on Random Logic Locking (RLL), which involves insertion of XOR/XNOR key-gates at arbitrary locations within a netlist [15]. While RLL established the foundational obfuscation paradigm, its efficacy was severely limited by a lack of structural protection, leading to predictable output corruption. The introduction of Boolean Satisfiability (SAT)-based attacks fundamentally shifted the threat model, as attackers could efficiently recover secret keys by identifying Distinguishing Input Patterns (DIPs) [8].

To counter SAT-based attacks, SARLock [9] and Anti-SAT [10] were developed as point function defenses that structurally constrain output corruption to a minimal number of input patterns, forcing SAT solvers to expend exponentially more iterations per key elimination. However, this SAT resilience comes at the cost of near-zero output corruptibility, leaving these schemes vulnerable to removal and bypass attacks that isolate and excise the obfuscation logic without key recovery [11].

Subsequent techniques addressed these limitations from different angles. FALL [18] leveraged fault-propagation analysis to maximize output corruption with minimal gate overhead, achieving approximately 50% Hamming distance between correct and incorrect outputs. SFLL [19] introduced provably secure locking through functional stripping, though its structural regularity exposed it to pattern-matching attacks. TALL [20] extended obfuscation into the temporal domain through timing-ambiguous

cells, while CLL [21] introduced deliberate feedback cycles to increase structural complexity. CCLL [22] combined distributed corruption with deterministic correction, offering resilience against structural analysis. Despite these advances, all classical techniques share static key-gate configurations that remain fixed post-fabrication, a fundamental limitation that leaves them exposed to long-term adversarial analysis [12].

3.2 Cryptographic Integration in Logic Locking

The integration of cryptographic primitives into hardware security flows addresses the inherent "static" nature of traditional Boolean-based logic locking. Block cipher-based approaches, most notably the Advanced Encryption Standard (AES)-based logic locking, offer strong theoretical resistance against differential and algebraic attacks. However, their practical deployment is constrained by substantial area and power requirements, multi-cycle latency penalties on timing-critical paths, and limited scalability for compact combinational circuits [26]. Furthermore, even block cipher-based schemes remain susceptible to long-term oracle-guided analysis if the underlying key remains static, undermining the fundamental security guarantee.

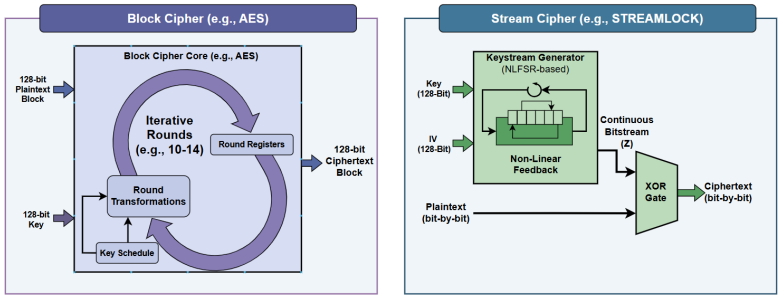


Fig. 1 Comparison of AES-based Block Cipher and NLFSR-based Stream Cipher Architectures

Stream ciphers have emerged as a more hardware-efficient cryptographic alternative, generating pseudorandom keystreams through compact NLFSR networks with significantly lower resource overhead than block ciphers [30]. As illustrated in Figure 1, the contrast between block cipher and stream cipher architectures is stark. AES requires iterative substitution-permutation rounds with dedicated round registers and key schedule logic, whereas the NLFSR-based stream cipher approach generates a continuous keystream through compact nonlinear feedback with minimal additional logic. Their inherent sensitivity to key and IV perturbations, where a single-bit deviation produces widespread, unpredictable output corruption, makes them particularly well-suited to logic locking applications. Prior work has explored stream cipher integration primarily in sequential and IoT-oriented designs [25], but comprehensive evaluation across standard combinational benchmarks and systematic comparison against both classical and block cipher-based locking schemes have remained absent from the literature. This work addresses that gap through STREAMLOCK, a lightweight 128-bit NLFSR-based locking mechanism evaluated across ISCAS'85 benchmarks of varying complexity [14].

3.3 Machine Learning for Hardware Security

The application of machine learning to hardware security has gained significant traction in recent years, driven by the growing complexity of IC designs and the limitations of manual, heuristic-driven protection strategies [1]. Early ML-based approaches focused primarily on attack-side applications, using neural networks and graph-based models to reverse engineer logic-locked circuits or predict key values from structural netlist features [12]. These developments highlighted a critical insight: if ML could be applied to break logic locking, it could equally be leveraged to strengthen it.

On the defense side, supervised learning has been explored for automating key-gate placement and selecting protection strategies based on circuit-level features. Ensemble methods, particularly Random Forest classifiers, have demonstrated strong generalization performance on small, imbalanced hardware security datasets due to their resilience to overfitting and natural feature importance ranking capabilities [16, 17]. However, existing ML-based frameworks largely address individual locking techniques in isolation, without considering the full architectural diversity of circuit types or integrating technique selection into a broader automated protection pipeline. This work addresses that limitation by training a Random Forest classifier on structural features extracted from combinational, sequential, and FSM-based benchmark circuits to automatically select the most appropriate locking technique from a portfolio of five classical schemes. This approach achieves consistent prediction accuracy across all circuit types [16, 17].

3.4 The Automation and Adaptability Gap

Despite the existence of these varied cryptographic and static primitives, a significant gap remains in the hardware security lifecycle: the lack of a systematic framework capable of automatically selecting the most appropriate locking strategy based on circuit-level architectural features. Existing solutions often rely on manual selection, utilizing rule-based logic or heuristic-driven, labor-intensive decision-making to determine key-gate placements. This manual bottleneck introduces critical vulnerabilities, as the microelectronics industry's drive toward smaller, more complex integrated circuits has made manual analysis exponentially difficult. Because over 90% of existing techniques are applied at the gate level as a post-synthesis step, designers are forced to manually analyze netlists. This process is not only time-consuming and prone to human error but fundamentally fails to scale with modern industrial benchmarks [1].

The necessity for a more efficient approach is further underscored by the susceptibility of the globalized "horizontal" business model. The inevitable integration of third-party IPs and outsourced fabrication makes the supply chain increasingly vulnerable to intellectual property piracy, reverse engineering, and the malicious insertion of hardware Trojans. While FPGA bitstream encryption offers an initial layer of defense, it remains vulnerable to side-channel attacks, probing, and modifications that allow adversaries to bypass protections [13]. Furthermore, standard logic locking implementations remain static after fabrication, leaving them open to long-term adversarial analysis once a bitstream is intercepted.

The unified pipeline presented in this work directly addresses all three limitations simultaneously, automating technique selection through a supervised Random Forest classifier, extending the protection space to include stream cipher-driven cryptographic locking via STREAMLOCK, and sustaining post-deployment security through Dynamic Partial Reconfiguration. This work represents the first integration of all three capabilities into a single, end-to-end hardware security framework evaluated across benchmark circuits of varying complexity.

4 Unified Pipeline Architecture

The integrity of hardware IPs across an untrusted supply chain demands a defense strategy that is simultaneously architecture-sensitive, cryptographically resilient, and operationally sustainable beyond the point of fabrication. No single locking technique satisfies all three requirements across the full diversity of modern IC designs, since combinational circuits, sequential machines, FSM-based controllers, and arithmetic datapaths each present distinct structural profiles that favor different protection mechanisms. Equally, the post-deployment security of a locked design cannot be treated as a solved problem at tape-out because field-deployed hardware remains exposed to evolving attack methodologies for its entire operational lifetime. The unified pipeline presented in this section addresses these requirements through four tightly coupled stages that collectively span the complete IC protection lifecycle. These stages comprise architecture-aware feature extraction and classifier training, ML-driven technique selection, key-gate implementation through either classical insertion or cryptographic stream cipher locking, and runtime reconfigurable post-deployment security via Dynamic Partial Reconfiguration (DPR).

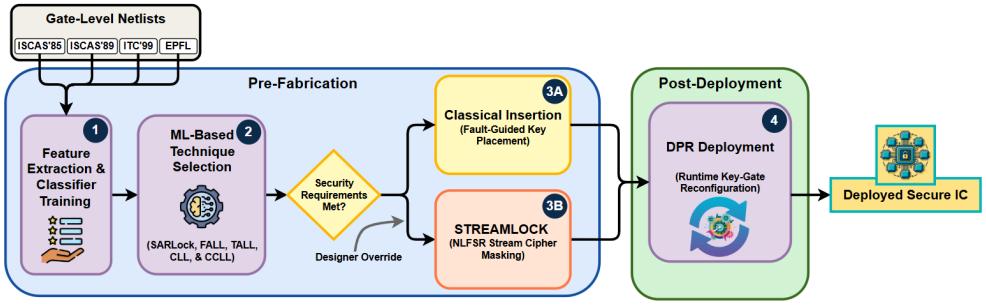


Fig. 2 End-to-end Unified IC Protection Pipeline Architecture

Figure 2 illustrates the end-to-end structure of the pipeline. In the first stage, structural and derived features are extracted from gate-level netlists and used to train a supervised Random Forest classifier. In the second stage, the trained classifier selects the most appropriate locking technique from a portfolio of five classical schemes based on each circuit's architectural profile. In the third stage, the selected technique is implemented either through fault-guided key-gate insertion for classical protection,

or through STREAMLOCK's nonlinear keystream integration for circuits requiring stronger cryptographic guarantees. In the fourth stage, the locked design is deployed on FPGA fabric with Dynamic Partial Reconfiguration enabled, allowing key-gate regions to be updated at runtime without system interruption.

4.1 Circuit Feature Extraction & Classifier Training

The first stage of the pipeline transforms gate-level circuit netlists into structured, information-rich feature vectors that drive all downstream classification decisions. Netlists are sourced from four established benchmark suites, ISCAS'85 [31], ISCAS'89 [32], ITC'99 [33], and EPFL [34], collectively spanning purely combinational designs, clocked sequential machines, FSM-based controllers, and arithmetic datapaths. This architectural diversity ensures that the trained classifier generalizes across the full range of circuit types encountered in real-world IC design flows. Because netlist formatting conventions differ substantially across benchmark families, three independent parsing modules are implemented, one each for combinational, sequential, and FSM-based circuit designs, to ensure consistent and accurate feature extraction regardless of netlist origin. From each parsed netlist, features are extracted and compiled into a unified feature vector. The first category, Primary Structural Features, comprises directly observable counts from the netlist:

- Logic gate counts, disaggregated by type (AND, OR, XOR, XNOR, NAND, NOR, and NOT), capture the compositional complexity of the combinational logic fabric.
- Sequential element counts, including D flip-flops and registers, quantify the degree of state-holding behavior in the design.
- FSM parameters, encompassing state count and transition complexity, characterize the control logic structure of FSM-based designs.
- Input/Output port complexity captures the primary interface dimensions of the design, reflecting the breadth of the observable input-output space available to an attacker.

The second category, Derived Security Metrics, comprises features computed from primary counts to capture each circuit's vulnerability profile across dimensions relevant to locking technique selection:

- Gate-to-Flip-Flop Ratio (GFFR): quantifies the balance between combinational and sequential logic, providing a normalized measure of how much of the circuit's complexity resides in stateless versus stateful elements.
- Sequential Versus Combinational Ratio (SVCR): measures the relative dominance of clocked logic over purely combinational paths, informing the suitability of timing-aware and cycle-based locking techniques.
- Approximate Circuit Path Depth (ACPD): estimates the length of the longest combinational logic cone from primary input to output, serving as a proxy for fault propagation reach and SAT-solver distinguishing input pattern complexity.
- Reconvergent Fan-In (RFI): quantifies the frequency with which diverged signal paths reconverge at a common gate, a structural property that amplifies the impact of key-gate insertion at reconvergence points.

- SAT Resilience Score (SR): characterizes the circuit's structural resistance to distinguishing input pattern generation, providing a pre-locking indicator of how aggressively SAT-based attacks are likely to exploit the design.
- Logic Cone Size (LCS): counts the number of primary inputs contributing to each circuit output, reflecting the breadth of the input space that must be explored to isolate correct key values.
- Fault Sensitivity Index (FS): quantifies the circuit's susceptibility to stuck-at fault propagation, directly informing the effectiveness of fault analysis-based key-gate placement strategies such as FALL.
- Interference Density (ID): captures the degree of signal interaction across the combinational logic network, characterizing how broadly a localized key-gate insertion will influence downstream circuit behavior.

Together, these features form the input representation supplied to the ML classifier, collectively encoding each circuit's structural complexity, sequential characteristics, and vulnerability profile across the dimensions most relevant to locking technique selection. To address the limited size of available benchmark datasets and improve model generalization, synthetic variants are generated by applying bounded perturbations of 5 to 10% to gate counts and sequential element parameters, preserving structural plausibility while expanding dataset coverage. Following augmentation, all numerical features are standardized to zero mean and unit variance prior to model training. The augmented dataset is first partitioned into an 80% training-validation pool and a 20% held-out test set via stratified sampling. The training-validation pool is further split 75%/25% into training and validation sets (also stratified by class), yielding an effective 60/20/20 split of the full dataset and approximately 600 training samples and 200 samples each for validation and testing.

Five supervised classifiers are trained and evaluated on this dataset. Random Forest, an ensemble of decision trees trained on bootstrap samples with random feature subsampling, is evaluated for its resilience to overfitting on small imbalanced datasets. XGBoost, a gradient boosting framework that incrementally builds decision trees with built-in regularization, is evaluated for its resistance to overfitting under regularized boosting. Decision Tree is included as an interpretable baseline. Logistic Regression is evaluated as a linear classifier to assess the degree of non-linearity in the feature space. Support Vector Machine with a linear kernel is evaluated for its margin-based classification capability. Among all five, Random Forest consistently achieves the highest test accuracy and strongest generalization, and is selected as the technique prediction engine for Stage 2 of the pipeline.

4.2 ML-Based Technique Selection

The second stage of the pipeline deploys the trained Random Forest classifier as an automated technique selection engine for previously unseen circuit designs. Given a new gate-level netlist, Stage 1 produces a structured feature vector that is passed to the trained model, which returns a predicted locking technique from the five-class target space: SARLock, FALL, TALL, CLL, and CCLL.

The assignment strategy is grounded in the architectural characteristics that distinguish each technique's domain of applicability. SARLock is assigned to circuits with low output complexity and minimal I/O fan-out, where SAT-solver resilience is the primary security objective and high output corruption is not a strict requirement. FALL is assigned to purely combinational circuits, leveraging its fault-propagation-guided insertion to maximize output corruption with minimal gate overhead. TALL is assigned to circuits with high logic depth or timing-sensitive pipelines, where its dual-domain obfuscation protects against both logical and temporal analysis. CLL is assigned to sequential and FSM-based circuits, where inherent architectural feedback paths allow cyclic locking cycles to be integrated without disrupting verified functionality. CCLL is assigned to hybrid circuits combining combinational and sequential logic, where its distributed corruption and deterministic correction mechanism provides broad coverage across heterogeneous designs, particularly favorable in resource-constrained sequential circuits where CLL's cyclic overhead is less justified.

For circuits where the static nature of classical key-gate insertion is deemed insufficient against oracle-guided or structural attacks, the pipeline supports a parallel cryptographic locking path. In this case, the design is routed to Stage 3's cryptographic implementation track rather than classical key-gate insertion. The decision between classical and cryptographic implementation is designer-driven, based on the security requirements and resource constraints of the target application.

4.3 Key-Gate Implementation

The third stage of the pipeline implements the locking strategy selected in Stage 2 through one of two parallel tracks: classical key-gate insertion, guided by fault-impact ranking to determine placement across all five candidate techniques, or cryptographic stream cipher-driven locking. The choice between these tracks determines both the security guarantees and the hardware overhead of the resulting locked design.

4.3.1 Classical Fault-Guided Insertion

For classical locking, key-gate insertion is performed using the HOPE parallel fault simulator [35], which estimates the fault impact at each circuit node by simulating input vectors and analyzing stuck-at fault propagation. Circuit nodes are ranked by their fault propagation potential, and key-gates, implemented as XOR or XNOR gates, are hierarchically inserted at the highest-ranked nodes. This fault-impact ranking serves as the shared placement heuristic across all five classical techniques identified in Stage 2. The technique selected determines which key-gate mechanism is instantiated at the ranked nodes, XOR/XNOR gates for FALL, a comparator network for SARLock, timing-ambiguous cells for TALL, feedback-path insertion for CLL, or distributed corruption-and-restore logic for CCLL, while the underlying node-selection methodology remains constant across all five. The locked netlist is then passed to Stage 4 for FPGA deployment.

4.3.2 STREAMLOCK Cryptographic Locking

For circuits routed to the cryptographic track, STREAMLOCK is deployed as the locking mechanism. Rather than embedding static key-dependent gates, STREAMLOCK locks the circuit through a continuously evolving keystream generated by a network of four deeply interconnected 32-bit Nonlinear Feedback Shift Registers (NLFSRs), as illustrated in Figure 3.

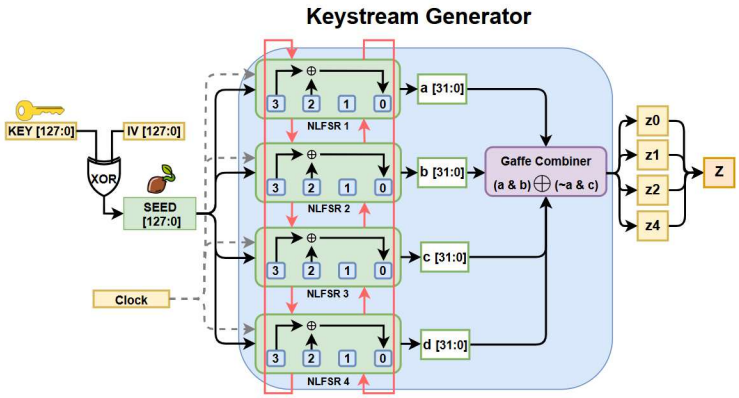


Fig. 3 128-bit NLFSR-Based STREAMLOCK Keystream Generator Architecture [14]

The framework is initialized by mixing a 128-bit secret key with a 128-bit initialization vector (IV) through a bitwise XOR operation, with nonlinear constants applied to each 32-bit segment to eliminate structural predictability and prevent degenerate all-zero initialization states. The resulting 128-bit seed is partitioned into four 32-bit segments, each used to initialize one NLFSR. Each register employs a unique primitive polynomial, ensuring no two registers share identical recurrence properties, and evolves its state through a nonlinear update function that combines polynomial feedback with bitwise logic operations.

A distinguishing architectural feature is the deep cross-coupling among the four NLFSRs, where each register's next-state computation incorporates selected state bits from all three remaining registers in addition to its own polynomial feedback. This cross-coupled structure ensures that any perturbation to one register's state propagates rapidly throughout the entire network, frustrating algebraic and correlation-based attacks by preventing the internal state from being factored into independent components.

After each NLFSR produces a 32-bit output, the four values are processed by a Geffe-inspired nonlinear output combiner to produce four parallel 32-bit keystream words (z0, z1, z2, and z3), concatenated to form the 128-bit keystream word Z on each clock cycle. Each output z_i is computed as:

$$z_i = (a \wedge b) \oplus (\neg a \wedge c) \tag{3}$$

with index offsets applied to diversify the four outputs and prevent any single NLFSR from disproportionately influencing the keystream.

Prior to producing valid keystream output, the system undergoes a 1152-cycle warm-up phase during which the NLFSRs evolve but the combiner output is suppressed. This ensures complete diffusion of the seed throughout the interconnected register network, eliminating statistical dependencies on the initial state. Once the warm-up counter expires, a ready signal is asserted and the 128-bit keystream words are applied to the circuit outputs via XOR gates, as illustrated for the C17 benchmark in Figure 4. Correct functionality is restored only when the authorized key initializes the NLFSR network into its proper internal state. Any deviation produces a mismatched state that propagates through the cross-coupled network, generating a divergent keystream that corrupts circuit outputs across a broad range of input patterns.

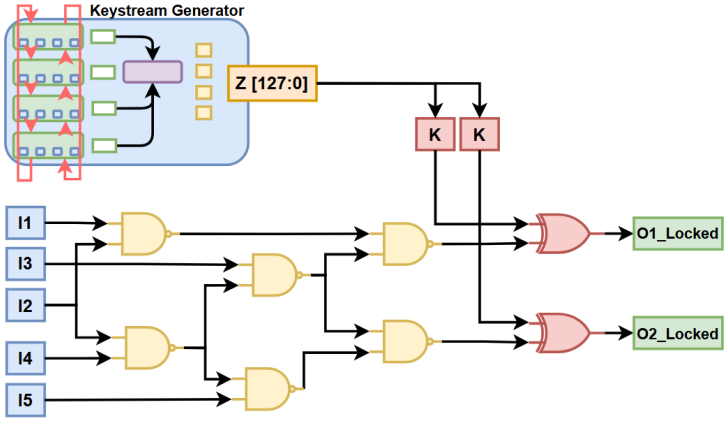


Fig. 4 STREAMLOCK Integration into the C17 Benchmark Circuit

4.4 Runtime Security via DPR

The integration of DPR into the logic-locking pipeline necessitates a specialized design methodology to ensure system stability and minimal performance impact during runtime updates. This workflow can be divided into a sequential process of synthesis, physical implementation, and bitstream generation.
Figure 5 provides a detailed architectural workflow for this process, illustrating the relationship between the Static Region and the Reconfigurable Partition. The standard implementation flow is modified to account for physical constraints required by the reconfigurable logic.

1. Design Synthesis & Partitioning: The initial phase involves defining the system architecture and isolating the logic that will undergo modification. This separates the design into a fixed static region and one or more Reconfigurable Partitions (RPs).

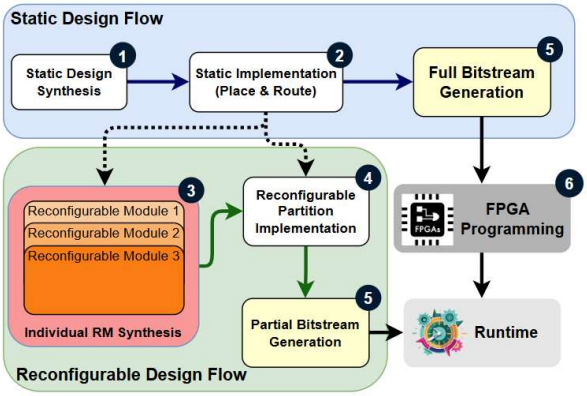


Fig. 5 DPR-based FPGA Implementation Workflow

2. Physical Implementation (Place & Route): During the implementation stage, fixed physical boundaries (Pblocks) are created on the FPGA fabric to contain the RPs. As seen in Figure 5, this defines the physical constraints within which multiple reconfigurable module variants can later be implemented.
3. Module Development: Each locking variant intended for runtime deployment, for example, an XOR gate configuration corresponding to key bit 0 and an XNOR gate configuration corresponding to key bit 1, is developed as an independent Reconfigurable Module (RM). Each RM undergoes individual synthesis to produce a self-contained implementation targeting the pre-defined RP boundaries.
4. Reconfigurable Partition Implementation: The synthesized RMs are mapped into the physical Pblock boundaries established during Place and Route. Each RM is independently placed and routed within its assigned partition, ensuring that the reconfigurable logic remains electrically isolated from the static region during runtime updates.
5. Bitstream Orchestration: Following implementation, two categories of bitstreams are generated. A full bitstream encapsulates the complete design, static region and initial RM configuration, for initial device programming. A partial bitstream is generated independently for each RM variant, containing only the configuration data required to update the corresponding reconfigurable partition.
6. FPGA Programming: The device is initially programmed using the full bitstream, establishing the static region and loading the default key-gate configuration. At runtime, key-gate reconfiguration is achieved by loading the appropriate partial bitstream through the FPGA configuration port, updating only the targeted partition without interrupting the operation of the static region.

This workflow enables selective, on-the-fly reconfiguration of key-gate regions with minimal disruption to the operational design. By decoupling the reconfiguration of individual partitions from the static region, DPR eliminates the need for full device reprogramming during key updates, a critical property for field-deployed systems where system downtime is operationally unacceptable. From a security perspective, the temporal variability introduced by runtime reconfiguration removes the fixed attack

surface that persists in conventionally deployed locked designs, a property expected to increase resistance to side-channel analysis and long-term statistical attacks that depend on a stable hardware configuration. Together, the four stages of the unified pipeline, architecture-aware feature extraction and classifier training, ML-driven technique selection, key-gate implementation, and DPR-based runtime security, collectively span the complete IC protection lifecycle from pre-fabrication design to post-deployment operation.

5 Experimental Results

This section presents the experimental evaluation of the unified pipeline across its constituent stages. Results are organized to reflect the pipeline's progression: beginning with the ML-based technique selection and its ground-truth resource validation, followed by STREAMLOCK cryptographic locking evaluation across multiple ISCAS'85 benchmark circuits, and concluding with a three-way comparative analysis among classical, stream cipher-based, and AES-based locking. DPR integration results are presented as the final pipeline stage, completing the end-to-end security evaluation.

5.1 Experimental Setup

All designs are implemented and evaluated targeting the programmable logic (PL) fabric of the Xilinx Zynq-7000 SoC on the Zedboard development platform using Vivado Design Suite. The Zynq-7000 PL fabric is based on the Artix-7 architecture, providing 53,200 Slice LUTs (Look-Up Tables, the primary combinational logic resources), 106,400 Slice Registers (flip-flops used for sequential storage), and 200 Bonded IOBs (Input/Output Blocks representing the physical interface pins) as the available resource pool. All synthesis and implementation runs use the default Vivado optimization strategy with no user-defined timing or area constraints applied, ensuring that reported resource utilization reflects the inherent overhead of each locking mechanism rather than constraint-driven optimization artifacts. Power consumption is extracted from Vivado post-implementation power analysis reports using default switching activity assumptions. Functional correctness of STREAMLOCK-locked designs is verified through cycle-accurate behavioral simulation in Vivado XSim.

5.2 ML-Based Technique Selection

The pipeline's automated technique-selection engine, the Random Forest classifier, is evaluated for dataset construction, classification performance, and ground-truth validation. Results first characterize generalization across augmentation levels, then quantify per-class accuracy, and finally validate predictions against expert-derived ground truth.

5.2.1 Dataset and Classifier Evaluation

The classifier is trained on gate-level netlists drawn from a diverse collection of benchmark circuits spanning combinational, sequential, FSM-based, and arithmetic design

types with varying structural complexities. Feature extraction produces a unified feature vector for each circuit. Ground truth labels are assigned through architectural inspection supported by prior literature, producing five technique classes, SARLock, FALL, TALL, CLL, and CCLL, as classification targets.

To address the limited size of available benchmark datasets and improve model generalization under class imbalance, synthetic variants are generated by applying bounded perturbations of 5 to 10% to gate counts and sequential element parameters, preserving structural plausibility while expanding dataset coverage. Empirical evaluation across augmentation levels from 5 to 20 variants per circuit establishes that 15 variants per original circuit provide the optimal balance between dataset diversity and classification accuracy. Beyond this threshold, further augmentation reduces dataset diversity and introduces redundant samples without measurable generalization benefit. The final augmented dataset yields approximately 600 training samples and 200 samples each for validation and held-out testing. As shown in Figure 6, the resulting class distribution shows Corrupt and Correct (300 samples) and Cyclic Logic Locking (285 samples) as the most represented classes and SARLock (75 samples) as the least, an intentional imbalance preserved to maintain model sensitivity to underrepresented techniques.

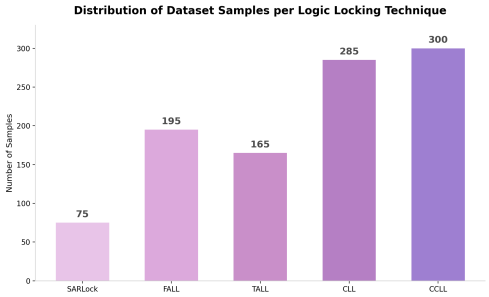


Fig. 6 Augmented Dataset Class Distribution

Five supervised classifiers are trained and evaluated on the augmented dataset: Random Forest, XGBoost, Decision Tree, Logistic Regression, and Support Vector Machine. As shown in Figure 7, training and test accuracy trends across all five models at varying augmentation levels reveal distinct generalization behaviors. Random Forest consistently achieves the highest test accuracy across all augmentation levels, maintaining a narrow gap between training and test performance, a strong indicator of resilience to overfitting on the small, imbalanced dataset. XGBoost demonstrates competitive training accuracy but exhibits progressive overfitting beyond the 12-variant threshold. Decision Tree similarly achieves near-perfect training accuracy but shows meaningful test performance degradation. Logistic Regression and SVM demonstrate consistently lower performance across all augmentation levels, reflecting their limited capacity to capture the non-linear decision boundaries between locking technique classes. Based on these results, Random Forest is selected as the technique-selection engine for the pipeline.

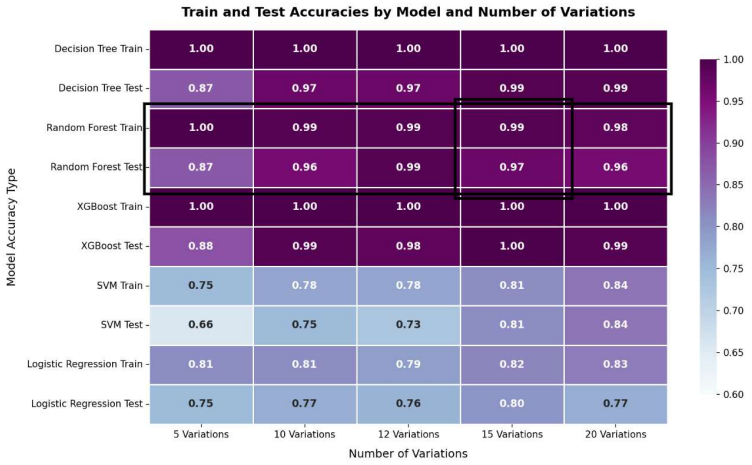


Fig. 7 Classifier Training and Test Accuracy Heatmap

5.2.2 Classification Performance

Table 1 summarizes the detailed classification performance of the Random Forest model at the 15-variant augmentation level. The model achieves a macro-average precision, recall, and F1-score of 0.96 across all five technique classes, confirming strong and balanced predictive performance even under class imbalance.

Table 1 Random Forest Classification Results with 15 Synthetic Variations

Class	Precision	Recall	F1-Score	Support
Corrupt and Correct	0.92	0.97	0.94	60
Cyclic Logic Locking	0.98	0.93	0.95	57
Fault Analysis	0.97	0.95	0.96	39
SARLock	1.00	1.00	1.00	15
Timing Aware	0.94	0.97	0.96	33
Macro Avg	0.96	0.96	0.96	204
Weighted Avg	0.96	0.96	0.96	204

SARLock achieves perfect precision, recall and F1-score of 1.00, with zero misclassifications in either direction. This reflects a distinct architectural signature despite being the most underrepresented class in training. Cyclic Logic Locking attains a precision of 0.98 and a recall of 0.93. Timing Aware attains a precision of 0.94 and a recall of 0.97. TALL is distinguished by sequential depth and FSM structure, while CLL is distinguished by feedback path density. Fault Analysis achieves a precision of 0.97 and a recall of 0.95, with two instances misclassified as Timing Aware, attributable to structural overlap between fault-propagation paths and timing-sensitive circuits. Corrupt and Correct attains the lowest precision at 0.92, with one instance each misclassified as Cyclic Logic Locking and Fault Analysis. This result is consistent with the architectural similarities among these techniques in hybrid circuit designs.

Figure 8 presents the confusion matrix for the Random Forest classifier, providing a visual summary of per-class prediction accuracy. The matrix confirms strong diagonal dominance across all five classes, with misclassifications concentrated at the boundaries between structurally similar techniques. The weighted average F1-score of 0.96 across 204 test samples establishes Random Forest as a reliable and consistent technique selection engine for the unified pipeline.

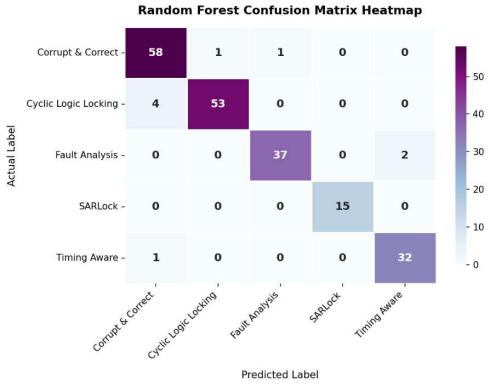


Fig. 8 Confusion Matrix for Random Forest Technique Classification

5.2.3 Feature Importance Analysis

Figure 9 illustrates the top 20 features ranked by their contribution to Random Forest predictions. Input count, FSM parameters, and AND gate count emerge as the strongest predictors of the optimal locking technique, followed by structural ratios such as GFFR and SVCR. This distribution is consistent with the architectural reasoning underlying the ground truth labeling strategy. TALL is driven primarily by sequential depth and FSM structure; CLL is driven by feedback path density, while FALL and CCLL are more strongly associated with combinational gate density and fault propagation potential.

Finer-grained structural factors including logic cone size, glitch sensitivity, and fan-in/fan-out contribute meaningfully to classification decisions, while basic element counts such as NAND gates and D flip-flops carry relatively lower predictive weight. The relatively low importance of raw gate counts compared to derived ratios confirms that the security-relevant metrics computed in Stage 1, GFFR, SVCR, ACPD, and FS in particular, capture architectural characteristics that raw structural counts alone cannot express.

The interpretability of the Random Forest feature importance ranking is a practically significant property of the framework. Unlike black-box classifiers, the feature importance scores provide a traceable, architecture-grounded justification for each technique assignment, directly supporting the design philosophy of the pipeline and enabling designers to audit and validate individual predictions against known circuit characteristics.

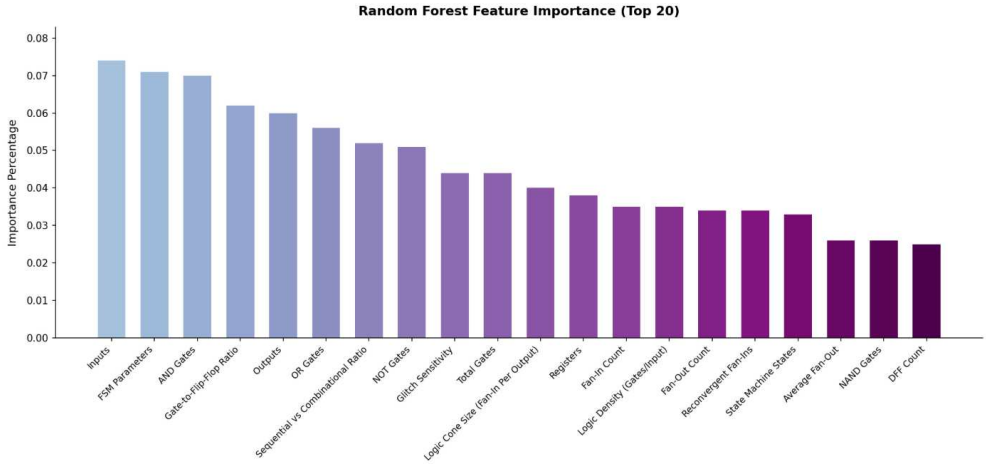


Fig. 9 Random Forest Feature Importance Rankings

5.2.4 Ground Truth Validation

To validate that ML-predicted technique assignments translate into practical hardware efficiency, all five candidate techniques are implemented across four benchmark circuits, C17, C432, S27, and S420, and their FPGA resource utilization is measured. Table 2 summarizes the ground truth technique assignments derived from architectural inspection, while Table 3 reports the corresponding resource metrics for both combinational and sequential circuits.

Table 2 Ground Truth Logic Locking Techniques for Combinational and Sequential Circuits

Circuit	Ground Truth Technique
C17	FALL (Fault Analysis Logic Locking)
C432	FALL (Fault Analysis Logic Locking)
S27	CCLL (Corrupt and Correct Logic Locking)
S420	CLL (Cyclic Logic Locking)

For the combinational C17 benchmark, FALL achieves the lowest area footprint at 3 LUTs (0.006%) while maintaining full output corruption coverage, validating the ML framework's prediction of FALL as the optimal technique for this circuit. SARLock consumes 5 LUTs for only partial corruption coverage, making it a less efficient choice for small combinational designs. In the larger C432 circuit, SARLock's resource demand rises sharply to 124 LUTs (0.23%) with heavy IOB utilization, while FALL maintains comparatively low overhead at 30 LUTs (0.06%), further reinforcing its suitability for combinational benchmarks.

For sequential circuits, resource scaling behavior differs. In the compact S27 benchmark, CCLL emerges as the most suitable technique, combining modest LUT

Table 3 Resource Utilization for Logic Locking Techniques in Combinational & Sequential Circuits

Technique	Key Bits	LUTs	IOBs	Key Bits	LUTs	IOBs	Registers
	C17 Circuit			**S27 Circuit**			
SARLock	3	5	10	5	4	9	3
FALL	5	3	12	5	5	11	3
CCLL	5	3	11	5	4	6	3
CLL	5	5	12	5	6	9	3
TALL	Not Applicable			5	5	11	3
	C432 Circuit			**S420 Circuit**			
SARLock	10	124	79	10	27	23	16
FALL	10	30	53	10	41	30	16
CCLL	10	55	50	10	41	30	16
CLL	10	47	49	10	39	30	16
TALL	Not Applicable			10	36	30	16

consumption with strong corruption coverage through its distributed fault injection and deterministic correction mechanism. In the larger FSM-based S420 circuit, CLL proves most effective, achieving high corruption coverage with 39 LUTs (0.07%) by leveraging the circuit's inherent feedback structure. While SARLock (27 LUTs) and TALL (36 LUTs) consume fewer resources, both offer weaker corruption coverage for this circuit, and CLL consumes fewer resources than both FALL and CCLL at 39 LUTs (0.07%) while providing stronger theoretical resilience against SAT-based attacks.

In all four cases, the ML-predicted technique matches the ground truth assignment derived from architectural analysis, confirming that the Random Forest classifier correctly identifies the most resource-efficient and security-appropriate locking strategy for each circuit type. This cross-validation between predicted and ground truth assignments establishes the reliability of the automated selection framework as a foundation for the subsequent implementation and evaluation stages.

5.3 STREAMLOCK Evaluation

5.3.1 Functional Correctness

Functional correctness of STREAMLOCK-based locking is verified through cycle-accurate behavioral simulation in Vivado XSim across five ISCAS'85 benchmark circuits. For each circuit, ten structurally diverse input vectors are applied following keystream warm-up, and the locked outputs are XORed with the frozen keystream word to recover the unlocked outputs. Correctness is confirmed when the recovered outputs match the golden circuit outputs for all ten input vectors.

Table 4 summarizes the verification results across all five circuits. All circuits achieve a perfect 10/10 match across all input vectors under the correct key, confirming 100% functional correctness of the STREAMLOCK integration. A particularly significant observation is that the frozen keystream word $z_{frozen} = 0x448e7d7a48c0fd6291c29d7a4043f11b$ is identical across all five circuits, a direct consequence of the fixed key and initialization vector applied consistently throughout. This deterministic and reproducible keystream behavior confirms that STREAMLOCK's

Table 4 STREAMLOCK Functional Verification Results Across ISCAS'85 Benchmark Circuits

Circuit	Gates	Output Count	Input Vectors	Correct Key Match	z_{frozen}
C17	6	2	10	YES (10/10)	`448e7d7a...4043f11b`
C432	160	7	10	YES (10/10)	`448e7d7a...4043f11b`
C880	383	26	10	YES (10/10)	`448e7d7a...4043f11b`
C1908	880	25	10	YES (10/10)	`448e7d7a...4043f11b`
C3540	1669	22	10	YES (10/10)	`448e7d7a...4043f11b`

initialization and warm-up sequence produces a stable, predictable output when the correct key is applied, regardless of the target circuit's structural complexity.

The consistency of z_{frozen} across circuits of vastly different sizes, from the 6-gate C17 to the 1,669-gate C3540, also validates the independence of the keystream generator from the locked circuit's combinational logic, confirming that STREAMLOCK's security properties are circuit-agnostic and do not degrade as circuit complexity scales.

5.3.2 Resource Utilization

Table 5 presents the post-implementation resource utilization and power consumption of STREAMLOCK-based designs across five ISCAS'85 benchmark circuits. Results are reported across three resource dimensions, Slice LUTs, Slice Registers, and Bonded IOBs, and three power components, total, logic, and I/O power.

LUT utilization scales moderately with circuit complexity, increasing from 141 LUTs for C17 to 371 LUTs for C3540. This scaling reflects the growing combinational logic contribution of each benchmark circuit layered on top of the fixed STREAMLOCK keystream generator overhead. Notably, the C17 baseline of 141 LUTs effectively represents the fixed overhead of the keystream generator itself, since C17 contains only 6 gates, confirming that the NLFSR network, warm-up counter, and output combiner collectively consume approximately 135-140 LUTs regardless of the target circuit's complexity.

Table 5 STREAMLOCK Resource Utilization and Power Consumption

Circuit	LUTs	IOBs	Registers	Tot. Power	Logic Power	I/O Power
C17	141	12	143	3.785	1.013	1.255
C432	188	48	148	9.299	1.304	5.055
C880	234	91	167	19.835	1.499	14.998
C1908	238	63	166	21.224	1.782	15.788
C3540	371	77	163	22.314	2.433	15.837

Slice Register count remains remarkably stable across all five circuits, ranging narrowly from 143 to 167 registers. This near-flat behavior directly reflects the fixed 128-bit NLFSR state, four 32-bit registers, that dominates the sequential resource footprint regardless of circuit size. The modest variation of 24 registers across a gate count range spanning more than two orders of magnitude confirms that STREAMLOCK's

sequential overhead scales independently of circuit complexity, a practically significant property for resource-constrained applications. Bonded IOB utilization scales with the number of primary inputs and outputs of each benchmark circuit rather than gate count, which explains why C880, with 60 inputs and 26 outputs, consumes the most IOBs at 91 despite having fewer gates than C1908 and C3540. Total power consumption rises from 3.785 W for C17 to 22.314 W for C3540, with I/O power the larger of the two reported components in every circuit, rising from 1.255 W for C17 to 15.837 W for C3540. Logic power, the component most directly attributable to the STREAMLOCK keystream generator, remains modest throughout, increasing from 1.013 W to 2.433 W across the full circuit range. Collectively, these results establish STREAMLOCK as a hardware-efficient cryptographic locking mechanism whose area and power overhead scales gracefully with circuit complexity, confirming its practical viability across a range of circuits.

5.4 Three-Way Comparative Analysis

A first systematic three-way comparison is conducted across all five ISCAS'85 benchmark circuits among classical fault-guided key-gate insertion, STREAMLOCK's stream cipher-based approach, and AES-based block cipher encryption. The comparison is grounded in Slice LUT utilization as the primary area metric, supplemented by a power consumption analysis comparing STREAMLOCK and AES-based locking across total, logic, and I/O power components.

Table 6 LUT Utilization Comparison: Classical (FALL), STREAMLOCK, and AES-Based Locking

Circuit	Gates	FALL LUTs	STREAMLOCK LUTs	AES LUTs
C17	6	3	141	1310
C432	160	30	188	1142
C880	383	72	234	1192
C1908	880	74	238	1194
C3540	1669	214	371	1325

Table 6 summarizes the LUT utilization of all three locking approaches across the five benchmark circuits. FALL, the representative classical technique selected by the ML framework for combinational circuits, demonstrates the lowest area footprint across all circuits, ranging from 3 LUTs for C17 to 214 LUTs for C3540. This modest overhead reflects the targeted nature of fault-guided key-gate insertion, which adds only a small number of XOR gates at high-impact nodes without introducing additional sequential logic beyond a compact key-loading shift register. However, the static nature of classical key-gate insertion means that the Boolean relationship between inputs, keys, and outputs remains fixed, a structural property that is systematically exploitable by SAT-based attacks and removal techniques.

STREAMLOCK occupies a clearly distinct middle tier, with LUT utilization ranging from 141 LUTs for C17 to 371 LUTs for C3540. The fixed lower bound of approximately 141 LUTs reflects the constant overhead of the NLFSR keystream

generator (four 32-bit interconnected registers, the nonlinear output combiner, and the warm-up counter), which is circuit-independent. The circuit-proportional component accounts for the growing combinational logic of larger benchmarks layered on top of this fixed cryptographic overhead. Across all five circuits, STREAMLOCK LUT utilization is 1.7× to 47× higher than FALL, reflecting the cost of introducing dynamic, time-varying keystream masking that fundamentally disrupts the static Boolean assumptions underlying SAT-based key recovery.

AES-based locking occupies the highest resource tier, with LUT utilization ranging from 1,142 LUTs for C432 to 1,325 LUTs for C3540, remaining effectively constant across the five circuits despite a gate count range spanning over two orders of magnitude. This near-constant behavior reflects the dominance of the fixed AES core overhead (S-Box substitution logic, round transformation networks, and key schedule), which dwarfs the combinational logic contribution of even the largest benchmark circuit. AES LUT utilization is 3.6× to 9.3× higher than STREAMLOCK and 6.2× to 437× higher than FALL across all circuits.

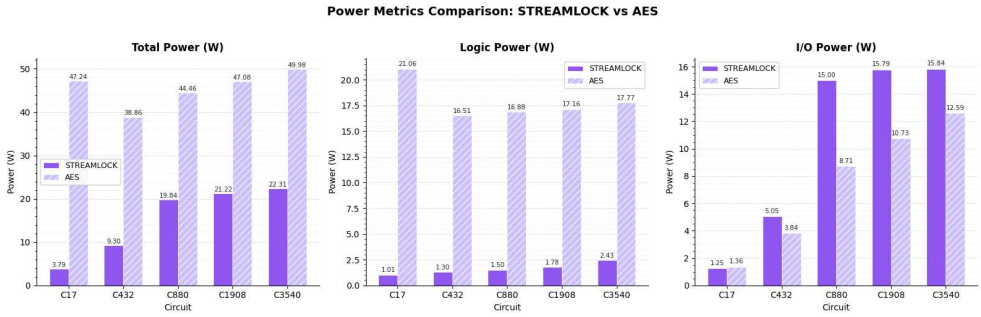


Fig. 10 Power Consumption Comparison Between STREAMLOCK and AES-Based Locking

Figure 10 presents a power consumption comparison between STREAMLOCK and AES-based locking across total, logic, and I/O power components respectively. The logic power separation is particularly pronounced. AES logic power ranges from 16.511 W to 21.059 W across all circuits, while STREAMLOCK logic power ranges from 1.013 W to 2.433 W, representing a 7.3× to 20.8× difference. This disparity directly reflects the combinational complexity of AES round transformations (SubBytes, ShiftRows, MixColumns, and AddRoundKey), which collectively consume substantially more switching power than STREAMLOCK's compact NLFSR feedback network. Total power consumption follows a similar trend, with AES reaching up to 49.982 W for C3540 compared to STREAMLOCK's 22.314 W. I/O power scales with circuit pin count for both approaches, though its contribution to total power differs: I/O power is the dominant component for STREAMLOCK across all circuits, while logic power remains the larger of the two reported components for AES across every circuit evaluated.

Together, the LUT utilization and power consumption results confirm that STREAMLOCK occupies a practically significant middle ground between classical

and AES-based locking. It delivers cryptographically stronger protection than static classical schemes through dynamic keystream evolution that prevents fixed Boolean analysis, while incurring substantially lower area and power overhead than AES-based alternatives across all evaluated circuit complexities.

5.5 DPR Integration

The fourth stage of the unified pipeline deploys the locked design on FPGA fabric with Dynamic Partial Reconfiguration enabled, providing a post-deployment security layer that sustains protection beyond the point of fabrication. DPR integration is demonstrated on the C17 benchmark circuit locked with Fault Analysis Logic Locking (FALL), as it represents the most resource-efficient classical locking configuration and provides a clean baseline for isolating DPR overhead.

The implementation partitions the design into a static region containing the primary circuit logic and five Reconfigurable Partitions (RPs), each encapsulating one of the five key-gates inserted by FALL. Each RP is initially programmed as an XOR gate corresponding to a key bit value of 0. At runtime, any RP can be independently reconfigured as an XNOR gate, corresponding to a key bit value of 1, by loading the corresponding partial bitstream through the FPGA configuration port, without halting or re-synthesizing the remainder of the design.

Table 7 Resource Utilization and Power Comparison: FALL versus FALL with DPR

Technique	Slice LUTs	Bonded IOBs	Tot. Power	Logic Power	I/O Power
FALL	2 (0.004%)	10	3.563	0.008	3.300
FALL + DPR	8 (0.015%)	10	3.774	0.036	3.446

Table 7 summarizes the resource and power overhead introduced by DPR integration. The inclusion of DPR increases LUT utilization from 2 to 8 Slice LUTs, an absolute increment of 0.011% of the available fabric. Total power consumption rises from 3.563 W to 3.774 W, representing a 5.92% increase driven primarily by the additional routing and configuration logic associated with the reconfigurable partition boundaries. Logic power increases from 0.008 W to 0.036 W, reflecting the overhead of the partial reconfiguration controller, while I/O power rises from 3.300 W to 3.446 W.

These results confirm that DPR integration introduces minimal hardware cost while substantially expanding the security posture of the deployed design. By enabling runtime key-gate updates without system interruption, DPR eliminates the static attack surface that persists in conventionally deployed locked designs, a capability that is particularly relevant for field-deployed systems in defense and critical infrastructure where periodic key rotation without downtime is operationally essential. The 0.011% LUT overhead establishes DPR as a practically viable post-deployment security enhancement for FPGA-based logic locking frameworks.

6 Conclusion

This paper presents a unified, four-stage machine learning-driven pipeline for IC logic locking that addresses the three fundamental limitations constraining existing hardware security frameworks: manual technique selection, static key-gate vulnerability, and the absence of post-deployment adaptability. By integrating architecture-aware ML-based technique selection, cryptographic stream cipher-driven locking via STREAMLOCK, and Dynamic Partial Reconfiguration into a single automated framework, the pipeline spans the complete IC protection lifecycle from pre-fabrication technique assignment to runtime key-gate reconfiguration.

The Random Forest classifier, trained on structural and security-relevant features extracted from diverse benchmark circuit families, achieves up to 97% prediction accuracy across five locking technique classes. Ground truth validation across four benchmark circuits confirms that ML-predicted assignments consistently match architecturally optimal technique choices, establishing the reliability of the automated selection layer.

STREAMLOCK's functional evaluation across five ISCAS'85 benchmark circuits confirms 100% correctness under the correct key, with a deterministic and reproducible keystream word consistent across all circuit implementations. The first systematic three-way comparison among classical, stream cipher-based, and AES-based locking demonstrates that STREAMLOCK achieves up to 9.3× lower LUT utilization and up to 20.8× lower logic power consumption than AES-based locking, while delivering cryptographically stronger protection than static classical schemes. DPR integration introduces only a 5.92% increase in total power consumption and a 0.011% increase in LUT utilization, confirming its viability as a low-cost post-deployment security enhancement.

Several directions remain open for future investigation. The current pipeline delegates the selection between classical and cryptographic locking tracks to the designer, and extending the ML framework to automatically predict the optimal protection family based on circuit-level security requirements and resource constraints represents a natural and high-impact next step. Integrating STREAMLOCK with DPR into a co-secured framework, where both the keystream parameters and the reconfigurable partition layout evolve dynamically at runtime, presents a promising avenue for stronger moving-target defense. Finally, extending the evaluation to include hardware Trojan detection and physical side-channel resistance characterization would broaden the security coverage of the pipeline.

Declarations

- Funding: The authors declare that no funds, grants, or other support were received.
- Competing interests: The authors have no competing interests.
- Ethics approval and consent to participate: Not applicable.
- Consent for publication: All authors have reviewed the final version of the manuscript and consent to its publication.
- Data availability: Not applicable.
- Materials availability: Not applicable.

- Code availability: Not applicable.
- Author contribution: Nahush Tambe conducted the research and worked on validation, investigation, and formal analysis of the project. Dr. Fareena Saqib supervised the entire research, writing and editing of the manuscript.

References

[1] Kamali, H.M., Azar, K.Z., Farahmandi, F., Tehranipoor, M.M.: Advances in logic locking: Past, present, and prospects. IACR Cryptol. ePrint Arch., 260 (2022)

[2] Tan, B., Karri, R., Limaye, N., Sengupta, A., Sinanoglu, O., Rahman, M.M., Bhunia, S., Duvalsaint, D., D., R., Blanton, Rezaei, A., Shen, Y., Zhou, H., Li, L., Orailoglu, A., Han, Z., Benedetti, A., Brignone, L., Yasin, M., Rajendran, J., Zuzak, M., Srivastava, A., Guin, U., Karfa, C., Basu, K., Menon, V.V., French, M., Song, P., Stellari, F., Nam, G.-J., Gadfort, P., Althoff, A., Tostenrude, J., Fazzari, S., Breckenfeld, E., Plaks, K.: Benchmarking at the Frontier of Hardware Security: Lessons from Logic Locking (2020). https://arxiv.org/abs/2006.06806

[3] Fan, J., Rushevich, G., Rusconi, G., Zhu, M., Dizon-Paradis, R., Forte, D.: Scalable IP Mimicry: End-to-End Deceptive IP Blending to Overcome Rectification and Scale Limitations of IP Camouflage (2026). https://arxiv.org/abs/2512.12061

[4] Masserova, E., Garg, D., Mai, K., Pileggi, L., Goyal, V., Parno, B.: Logic Locking - Connecting Theory and Practice. Cryptology ePrint Archive, Paper 2022/545 (2022). https://eprint.iacr.org/2022/545

[5] Subbiah, K., Chinnathevar, S.: A survey on logic-locking characteristics and attacks. Journal of The Institution of Engineers (India): Series B **105**(4), 1073–1087 (2024) https://doi.org/10.1007/s40031-024-01017-y

[6] Reimann, L.M., Rezunov, E., Germek, D., Collini, L., Pilato, C., Karri, R., Leupers, R.: The impact of logic locking on confidentiality: An automated evaluation. In: 2025 26th International Symposium on Quality Electronic Design (ISQED), pp. 1–8 (2025). https://doi.org/10.1109/ISQED65160.2025.11014457

[7] Engels, S., Hoffmann, M., Paar, C.: A critical view on the real-world security of logic locking. Journal of Cryptographic Engineering **12**, 229–244 (2022) https://doi.org/10.1007/s13389-022-00294-x

[8] Subramanyan, P., Ray, S., Malik, S.: Evaluating the security of logic encryption algorithms. In: 2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 137–143 (2015). https://doi.org/10.1109/HST.2015.7140252

[9] Yasin, M., Mazumdar, B., Rajendran, J.J.V., Sinanoglu, O.: Sarlock: Sat attack resistant logic locking. In: 2016 IEEE International Symposium on Hardware

Oriented Security and Trust (HOST), pp. 236–241 (2016). https://doi.org/10.1109/HST.2016.7495588

[10] Xie, Y., Srivastava, A.: Anti-sat: Mitigating sat attack on logic locking. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **38**(2), 199–207 (2019) https://doi.org/10.1109/TCAD.2018.2801220

[11] Yasin, M., Mazumdar, B., Sinanoglu, O., Rajendran, J.: Removal attacks on logic locking and camouflaging techniques. IEEE Transactions on Emerging Topics in Computing **8**(2), 517–532 (2020) https://doi.org/10.1109/TETC.2017.2740364

[12] Gandhi, J., Shekhawat, D., Santosh, M., Pandey, J.G.: Logic locking for ip security: A comprehensive analysis on challenges, techniques, and trends. Comput. Secur. **129**(C) (2023) https://doi.org/10.1016/j.cose.2023.103196

[13] Puschner, E., Ender, M., Becker, S., Paar, C.: Patching fpgas: The security implications of bitstream modifications. In: Proceedings of the 2024 Workshop on Attacks and Solutions in Hardware Security. CCS '24, pp. 89–99. ACM, ??? (2024). https://doi.org/10.1145/3689939.3695779 . http://dx.doi.org/10.1145/3689939.3695779

[14] Tambe, N., Aklekar, D., Lodge, N., Chadalavada, V., Saqib, F.: Streamlock: Stream cipher-enabled cryptographic logic locking. In: 2026 27th International Symposium on Quality Electronic Design (ISQED), pp. 1–7 (2026). https://doi.org/10.1109/ISQED69900.2026.11534769

[15] Roy, J.A., Koushanfar, F., Markov, I.L.: Epic: Ending piracy of integrated circuits. In: 2008 Design, Automation and Test in Europe, pp. 1069–1074 (2008). https://doi.org/10.1109/DATE.2008.4484823

[16] Tambe, N., Lodge, N., Chadalavada, V., Saqib, F.: Implementation of machine learning-based dynamic logic locking. In: 2025 IEEE Physical Assurance and Inspection of Electronics (PAINE), pp. 1–7 (2025). https://doi.org/10.1109/PAINE66113.2025.11320174

[17] Tambe, N., Lodge, N., Aklekar, D., Saqib, F.: Architecture-aware automated logic locking: A machine learning (ml)-based implementation. In: 2025 IEEE 22nd International Conference on Smart Communities: Improving Quality of Life Using AI, Robotics and IoT (HONET), pp. 207–212 (2025). https://doi.org/10.1109/HONET67928.2025.11318461

[18] Rajendran, J., Zhang, H., Zhang, C., Rose, G.S., Pino, Y., Sinanoglu, O., Karri, R.: Fault analysis-based logic encryption. IEEE Transactions on Computers **64**(2), 410–424 (2015) https://doi.org/10.1109/TC.2013.193

[19] Yasin, M., Sengupta, A., Nabeel, M.T., Ashraf, M., Rajendran, J.J., Sinanoglu,

O.: Provably-secure logic locking: From theory to practice. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. CCS '17, pp. 1601–1618. Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3133985 . https://doi.org/10.1145/3133956.3133985

[20] Talukdar, J., Paik, W.-H., Ortega, E., Chakrabarty, K.: Alt-lock: Logic and timing ambiguity-based ip obfuscation against reverse engineering. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **32**(8), 1535–1548 (2024) https://doi.org/10.1109/TVLSI.2024.3411033

[21] Chen, P.-P., Yang, X.-M., He, Y.-C., Chen, Y.-C., Li, Y.-T., Wang, C.-Y.: Looplock 3.0: A robust cyclic logic locking approach. In: 2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 594–599 (2024). https://doi.org/10.1109/ASP-DAC58780.2024.10473877

[22] Aksoy, L., Yasin, M., Pagliarini, S.: CAC 2.0: A Corrupt and Correct Logic Locking Technique Resilient to Structural Analysis Attacks (2024). https://arxiv.org/abs/2401.07142

[23] Lie, W., Feng-yan, W.: Dynamic partial reconfiguration in fpgas. In: 2009 Third International Symposium on Intelligent Information Technology Application, vol. 2, pp. 445–448 (2009). https://doi.org/10.1109/IITA.2009.334

[24] Kizheppatt, V., Fahmy, S.A.: Fpga dynamic and partial reconfiguration: A survey of architectures, methods, and applications. ACM Computing Surveys **51**(4), 1–39 (2018) https://doi.org/10.1145/3193827

[25] Khan, M., Dagenborg, H., Johansen, D.: Performance evaluation of lightweight stream ciphers for real-time video feed encryption on arm processor. Future Internet **16**(8) (2024) https://doi.org/10.3390/fi16080261

[26] Chhabra, S., Lata, K.: Design and analysis of logic encryption based 128-bit aes algorithm: A case study. In: 2018 15th IEEE India Council International Conference (INDICON), pp. 1–6 (2018). https://doi.org/10.1109/INDICON45594.2018.8987098

[27] De Cannière, C., Preneel, B.: Trivium. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs: The eSTREAM Finalists vol. 4986, pp. 244–266. Springer, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68351-3_18 . https://doi.org/10.1007/978-3-540-68351-3_18

[28] Hell, M., Johansson, T., Meier, W.: Grain: a stream cipher for constrained environments. Int. J. Wire. Mob. Comput. **2**(1), 86–93 (2007) https://doi.org/10.1504/IJWMC.2007.013798

[29] Babbage, S., Dodd, M.: In: Robshaw, M., Billet, O. (eds.) The MICKEY Stream

Ciphers, pp. 191–209. Springer, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68351-3_15 . https://doi.org/10.1007/978-3-540-68351-3_15

[30] Dubrova, E.: How to speed-up your nlfsr-based stream cipher. In: 2009 Design, Automation & Test in Europe Conference & Exhibition, pp. 878–881 (2009). https://doi.org/10.1109/DATE.2009.5090786

[31] Brglez, F., Fujiwara, H.: A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS), 663–698 (1985)

[32] Malagi, S.: ISCAS 85/89, ITC 99, and Other Benchmarks for Logic Synthesis and Testing. https://github.com/santoshsmalagi/Benchmarks. Accessed: 2025-07-09

[33] Corno, F., Sonza Reorda, M., Squillero, G.: Rt-level itc'99 benchmarks and first atpg results. IEEE Design & Test of Computers **17**(3), 44–53 (2000) https://doi.org/10.1109/54.867894

[34] Amaru, L.G., Gaillardon, P.-E., Micheli, G.D.: The epfl combinational benchmark suite. In: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1257–1262. IEEE, ??? (2015). https://doi.org/10.7873/DATE.2015.109 . https://github.com/lsils/Benchmarks

[35] Lee, H.K., Ha, D.S.: Hope: an efficient parallel fault simulator for synchronous sequential circuits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **15**(9), 1048–1058 (1996) https://doi.org/10.1109/43.536711