TroPUF: Evaluating Hardware Trojan Insertion in Delay-Based Physical Unclonable Functions

Marissa Marcarelli
Computer Engineering & Computer Science Department
California State University Long Beach
Long Beach, CA, USA
marissa.marcarelli01@student.csulb.edu

Amin Rezaei
Computer Engineering & Computer Science Department
California State University Long Beach
Long Beach, CA, USA
amin.rezaei@csulb.edu

***Abstract*—Delay-based Physical Unclonable Functions (PUFs) are commonly used for device authentication and key generation due to the fact that they rely on manufacturing induced delay variations. However, these same variations make PUFs inherently non-deterministic, which can allow malicious logic to blend in with normal circuit behavior. As a result, the act of embedding hardware Trojans directly inside the PUF primitive presents a unique security risk that is not yet well understood. This work presents a unified simulation framework for evaluating stealthy hardware Trojan insertion across multiple delay-based PUF architectures and Trojan types. Functional metrics, hardware overhead, and resistance to machine learning modeling are assessed in parallel. Results show that dormant Trojans preserve expected PUF behavior, structural characteristics, and modeling resistance. Detectable degradation appears only after activation, indicating that conventional validation techniques fail to identify embedded Trojans prior to payload execution. These findings expose a gap in current PUF security assumptions, and highlight the need to evaluate PUFs and hardware Trojans as a coupled security problem.**

***Index Terms*—Physical Unclonable Function, Delay Variation, Hardware Trojan, Modeling Attack, Hardware Security Primitive**

I. Introduction

With the increasing reliance on hardware-based authentication and secure key storage, guaranteeing the trustworthiness of security primitives has become progressively important [5].

Physical Unclonable Functions (PUFs) [10] are hardware security primitives that exploit manufacturing-induced physical variation to generate responses that are difficult to clone and unique between devices [10], [11], [28]. Their quality is commonly evaluated using metrics such as uniqueness, uniformity, randomness, and reliability [14], [20], which measure inter-device distinctiveness and intra-device stability. Satisfying these properties is generally considered a foundational requirement for security application use [27].

Among different PUFs, delay-based PUFs are one of the most widely studied due to their simplicity and scalability [28], [36]. They derive responses by relying on small timing differences along paths designed to be symmetric, where manufacturing-induced variation produces imbalances [24].

At the same time, hardware Trojans remain a serious threat in integrated circuits [29], [39]. A Trojan consists of a rare trigger and malicious payload, and is designed to remain dormant under normal operation [15], [16]. Most detection techniques assume deterministic logic and attempt to identify structural, functional, or side-channel anomalies [4], [13]. PUFs challenge these assumptions because PUF behavior is intentionally nondeterministic [1]. Since variability is expected, small irregularities introduced by a Trojan may appear indistinguishable from natural PUF behavior, allowing it to be hidden by the primitive itself.

Machine Learning (ML) further complicates this issue. Modeling attacks have demonstrated that delay-based PUF responses can be approximated with high accuracy using supervised learning techniques [23], [31]. While modeling resistance is often treated as evidence of PUF strength [8], [32], a Trojan-infected PUF may still preserve modeling resistance under conventional evaluation methods.

Contrary to common evaluation practices, which treat PUF security and hardware Trojan detection separately, the interaction between the two has not yet been studied in depth. Satisfying functional metrics and modeling resistance does not guarantee integrity when malicious logic is embedded directly within the PUF primitive. In this paper, we present for the first time a unified framework called **TroPUF** that analyzes hardware Trojan insertion directly in delay based PUFs. Our contributions, as seen in Fig. 1 are as follows:

- Developing a unified framework for evaluating hardware Trojan insertion across multiple delay-based PUFs.
- Analyzing functional metrics and hardware overhead of clean and Trojan-infected PUF implementations.
- Evaluating modeling resistance of Trojan-infected PUFs and demonstrating that satisfying conventional PUF metrics and modeling resistance does not guarantee security.

A. Related Works

1) Delay-Based PUFs: Silicon physical random functions were introduced in earlier work [11] and later extended for authentication and key generation [12], [28]. Delay-based PUFs, particularly the Arbiter PUF, exploit race conditions along symmetric delay paths to generate responses from manufacturing variation [28], [36]. Architectural extensions include Butterfly [17], Ring Oscillator [6], [25], XOR [40], Feed-forward and Interpose [2], [22], and cyclic designs such as CycPUF [7]. However, increased structural complexity does not inherently improve security [3], [38]. Evaluation typically relies on statistical metrics including uniqueness, reliability, and uniformity [14], [20], [27], assuming trusted implementation.

2) Modeling Attacks on PUFs: It was demonstrated that Arbiter PUFs can be approximated using linear additive delay models trained on challenge-response pairs [23]. Increasing architectural complexity does not inherently prevent learnability [31]. This led to the development of ML-resistant designs [32], modified arbiter structures [8], and robustness evaluation tools such as PUFmeter [9]. Modeling resistance, however, is typically evaluated under benign structural assumptions.

3) Hardware Trojans: Hardware Trojans are typically categorized based on their trigger mechanisms and payload characteristics [16], [29], [39]. Prior work has shown that stealthy insertion is achievable with minimal hardware footprint [15], including designs that use counter-based triggers [18], sequential activation strategies [35], and scalable mechanisms that exploit rare trigger conditions [19].

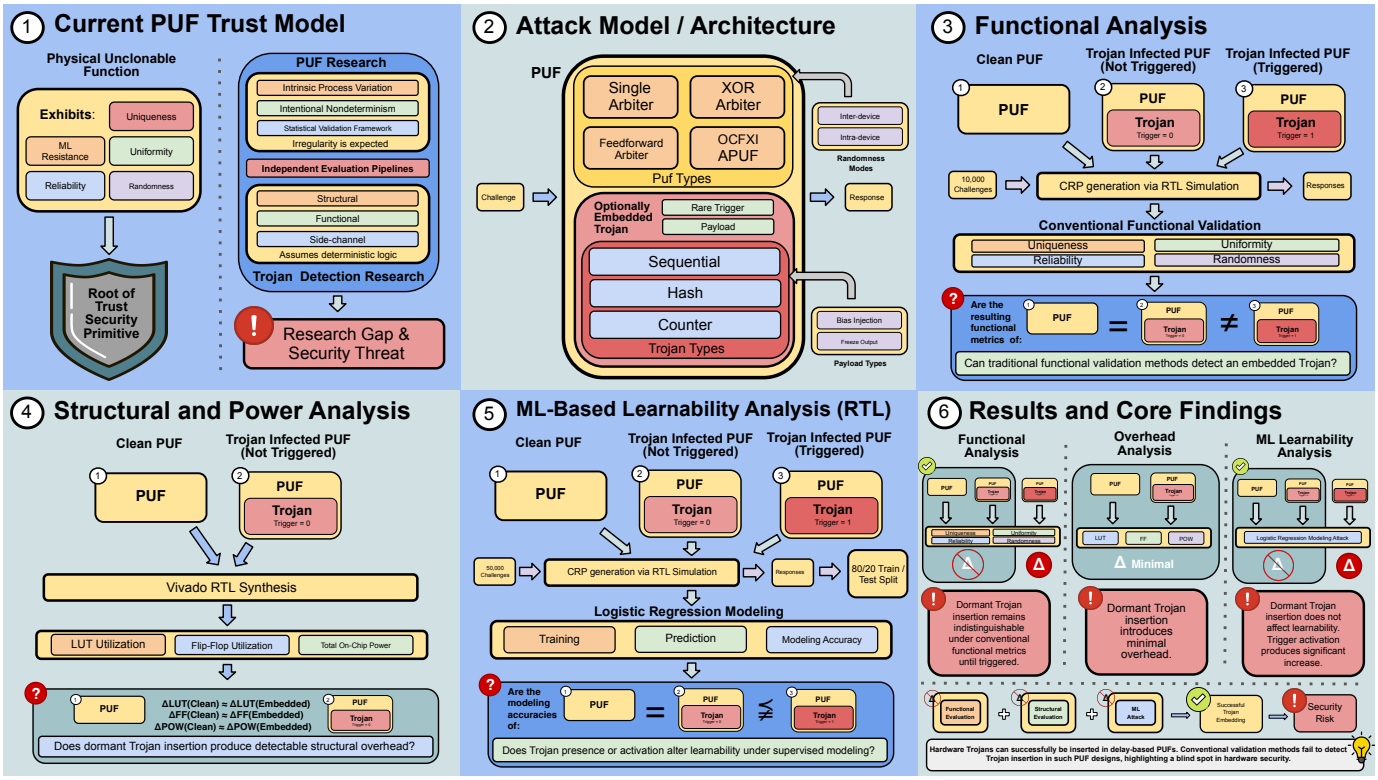

Fig. 1. TroPUF Framework.

4) Trojan Detection: Detection approaches typically rely on structural, functional, and side-channel analysis [4]. Functional methods attempt rare trigger activation [19], while side-channel techniques analyze power and delay deviations [30]. ML has also been applied to side-channel data to improve detection accuracy [13]. However, these methods assume deterministic circuit behavior. Recent work has further explored advanced detection and strategies, including risk-aware and explainable frameworks that aim to improve coverage and interpretability [33]. Additionally, multimodal deep learning approaches have been proposed to enhance detection performance under uncertainty by leveraging multiple data sources [34]. Complementary efforts have also investigated runtime mitigation techniques for addressing hardware Trojans in deployed systems, particularly in the context of logic-locked circuits [21].

5) PUFs and Trojan Interaction: PUFs violate deterministic assumptions due to intrinsic process variation [1], complicating anomaly-based detection. Prior work has used PUFs within broader security frameworks [1] and examined PUF-assisted Trojan attacks [26]. However, these studies treat PUFs primarily as detection or authentication mechanisms. Prior research has not specifically analyzed the security implications of embedding hardware Trojans directly within delay-based PUF primitives, nor evaluated its effects.

B. Motivation

Delay-based PUF research emphasizes statistical quality and modeling resistance [14], [20], while Trojan research focuses on stealth and anomaly detection under deterministic assumptions [29], [39]. These domains remain largely independent. Whether a Trojan-infected delay-based PUF can simultaneously satisfy functional metrics, preserve modeling resistance, and evade detection remains unexplored. The assumption of trustworthiness does not hold when the security primitive itself serves as the Trojan host. Since delay-based PUFs often serve as root-of-trust primitives, compromise of the PUF also compromises higher level security mechanisms that depend on it. This then enables attacks across entire platforms or systems. This gap motivates and clearly highlights the need for the unified evaluation framework proposed in this work.

C. Threat Model

We consider an adversary that embeds a hardware Trojan within a delay-based PUF during design or fabrication. The Trojan consists of a rare trigger and dormant payload, remaining inactive under normal operation. Prior to activation, the PUF preserves expected functional and structural behavior and modeling resistance. Upon activation, the payload alters PUF behavior by exploiting intrinsic variability. The defender evaluates the PUF using conventional validation techniques. No trusted reference is assumed, and PUF variability is expected.

II. Preliminaries

This section presents the preliminaries relevant to this work, including delay-based PUF architectures, standard evaluation metrics, synthesis-level analysis, and ML-based modeling attacks.

A. Delay Based PUFs

Delay-based PUFs derive responses from small differences in signal propagation time along nominally symmetric circuit paths. These differences come from manufacturing variation at the transistor and interconnect level, producing device specific imbalances [11], [24]. Since the delays are physically embedded, the resulting responses are difficult to replicate across devices. The behavior can be modeled as a linear function of the applied challenge, allowing scalable and lightweight implementations. In addition, due to their

low area overhead and CMOS compatibility, delay-based PUFs are widely used in authentication and root-of-trust applications [10], [27].

A classic example is the Arbiter PUF, where two signals race through multiplexed delay chains controlled by a challenge vector, and an arbiter outputs a bit based on arrival time [28], [36]. Other constructions, such as Ring Oscillator and XOR Arbiter PUFs, exploit frequency differences or combine multiple delay paths to increase complexity [25], [40].

B. PUF Metrics

Functional metrics represent the primary means by which PUF designs are traditionally validated and trusted [14], [20], [27]. By applying the same metrics to both clean and Trojan infected designs, we can determine if standard validation techniques are sufficient to detect malicious modification. In the following formulas, N represents the number of PUF instances, n is the response length in bits, m is the number of evaluated responses, and s is the number of repeated measurements. R_i and R_j are response vectors from different PUF instances, R_i^* represents repeated evaluations of the same PUF instance under identical challenges, and $p(x)$ denotes the probability of observing the response bit x.

For reference, the Hamming Distance (HD) between two response vectors R_1 and R_2 is defined as:

$$\mathrm{HD}(R_1, R_2) = \sum_{i=1}^{n} \left(R_1[i] \oplus R_2[i]\right), \tag{1}$$

The Hamming Weight (HW) of a response vector R is given by:

$$\mathrm{HW}(R) = \sum_{i=1}^{n} R[i]. \tag{2}$$

1) Uniqueness: Different PUF instances with the same design should ideally produce distinct responses to the same challenge. Uniqueness measures this difference, typically defined as the normalized inter-chip HD. The ideal uniqueness that a PUF design can achieve is 50%.

$$\text{Uniqueness} = \frac{2}{N(N-1)} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{HD(R_i, R_j)}{n} \times 100\% \tag{3}$$

2) Reliability: Reliability measures the consistency of a PUF response when the same challenge is applied multiple times. It quantifies the intra-device stability of the generated responses under varying conditions. An ideal PUF achieves a reliability score of 100%.

$$\text{Reliability} = \left(1 - \frac{1}{s} \sum_{i=1}^{s} \frac{HD(R, R_i^*)}{n}\right) \times 100\% \tag{4}$$

3) Uniformity: Uniformity measures whether the "1"s and "0"s in PUF responses are evenly balanced. Ideally, a PUF should produce equal numbers of both values across all challenges, indicating no inherent bias. An ideal PUF achieves a uniformity score of 50%.

$$\text{Uniformity} = \frac{1}{m} \sum_{i=1}^{m} \frac{HW(R_i)}{n} \times 100\%, \tag{5}$$

4) Randomness: Randomness captures the statistical unpredictability of PUF responses. Ideally, output bits should exhibit no discernible patterns or correlations, even when challenges are closely related. An ideal PUF will reach a randomness score of 100%.

$$H(X) = -\sum_{x \in \{0,1\}} p(x) \log_2 p(x) \times 100\%, \tag{6}$$

C. Synthesis Analysis

FPGA synthesis level metrics are evaluated to provide a relative estimate of the structural overhead introduced by Trojan insertion under consistent synthesis conditions [15], [29]. These metrics capture changes in resource utilization and power characteristics that may result from embedded malicious logic, including Lookup Table (LUT) utilization, Flip Flop (FF) utilization, and power consumption.

D. ML Modeling Attacks

ML attacks approximate the functional relationship between challenges and responses using supervised models trained on collected Challenge–Response Pairs (CRPs). For delay-based PUFs such as the Arbiter PUF, responses can be represented using an additive linear delay model, where each challenge corresponds to a feature vector and the output is determined by the sign of a weighted sum [23]. Algorithms including logistic regression and support vector machines can approximate this model given sufficient CRPs [23], [31]. Modeling resistance is commonly evaluated by measuring prediction accuracy under such attacks [9], [32]. In this work, prediction accuracy is used as an evaluation metric to assess whether Trojan insertion alters learnability without disrupting functional correctness.

III. System Architecture and Implementation

This section describes the proposed unified evaluation framework called **TroPUF**, which analyzes delay-based PUFs under both clean and Trojan-infected configurations. **TroPUF** includes various delay-based PUF designs, hardware Trojan models, and payload types to evaluate how malicious logic interacts with PUF behavior across complex implementations.

A. PUF Implementations

1) Single Arbiter PUF (SA-PUF): As seen in Fig. 2 the SA-PUF serves as a baseline architecture. It consists of N cascaded delay stages controlled by challenge bits. A start pulse launches complementary signals into symmetric multiplexed delay paths, where each stage conditionally swaps the signal ordering based on the applied challenge bit. The final race is resolved by an arbiter to produce a response bit [14], [28]. Its linear additive delay model makes it vulnerable to modeling attacks [3], [23], providing a reference point for evaluating how increased complexity affects Trojan detectability and stability.

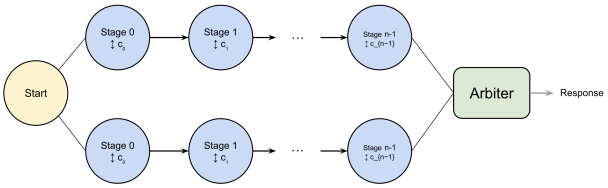

Fig. 2. Single Arbiter PUF architecture.

2) Feedforward Arbiter PUF (FA-PUF): The FA-PUF, shown in Fig. 3, introduces stage level dependency by re-injecting an intermediate race result from an early delay stage into later stages through XOR challenge perturbation. This feedforward signal modifies the effective challenge vector applied to subsequent multiplexers, breaking linear separability of the additive delay model. Feedforward structures improve modeling resistance [8], [14], although added complexity does not eliminate vulnerability [31]. The FA-PUF serves as an intermediate complexity design for evaluating how structural feedback influences Trojan behavior.

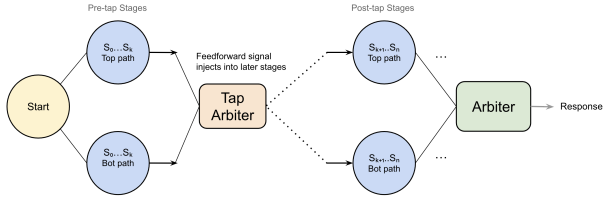

Fig. 3. Feedforward Arbiter PUF architecture.

3) XOR Arbiter PUF (XA-PUF): The XA-PUF as seen in Fig. 4, instantiates multiple independent arbiter delay chains in parallel and combines their response bits using bitwise XOR operation, producing a parity-based response. This implementation introduces nonlinear interaction between multiple delay races, and expands the hypothesis space required for modeling [3], [40], although large-scale attacks remain effective under realistic conditions [31], [37]. From a Trojan perspective, parallel composition increases activity and logic footprint, allowing evaluation of how aggregation affects detectability.

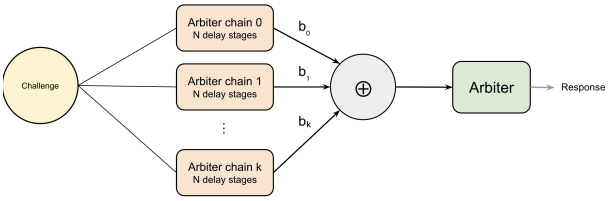

Fig. 4. XOR Arbiter PUF architecture.

4) Obfuscated Composite Feedforward XOR Interpose Arbiter PUF (OA-PUF): The OA-PUF pictured in Fig. 5, is the most complex architecture in the framework. An upper feedforward PUF bank generates an intermediate response that is interposed into the challenge vector of a lower XOR based bank, forming a hierarchical dependency similar to interpose constructions proposed for modeling resistance [22], [38]. This structure significantly increases internal dependency depth and switching activity, making it useful for evaluating whether architectural complexity conceals malicious logic.

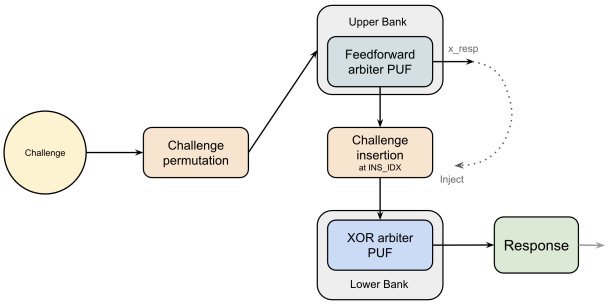

Fig. 5. Obfuscated Composite Feedforward XOR Interpose Arbiter PUF architecture.

B. Trojans and Triggers

The Trojans evaluated in this work are manually designed for controlled experimentation. Such designs are based on commonly studied trigger mechanisms described in prior hardware Trojan literature, including sequential, counter, and combinational triggers.

1) Sequential: This Trojan triggers only after a predefined ordered sequence of events. A finite-state machine advances when three specific challenges appear in the correct order, and resets upon deviation [35]. Once the final state is reached, activation is latched and the payload executes. Since activation depends on sequences rather than isolated inputs, accidental triggering is unlikely. This trigger evaluates how rare activation interacts with architectural complexity.

2) Counter: This Trojan activates after prolonged use by counting valid events. In this case, each start pulse increments an internal counter, and activation occurs once a predefined threshold is reached [18], [29]. Because activation depends only on event count, it avoids direct statistical correlation with challenge content. The counter trigger enables evaluation of long latency activation, and whether accumulated usage produces detectable anomalies.

3) Hash: This Trojan activates when an XOR based hash of the input challenge matches a predefined target value. Rather than monitoring raw challenge bits, the trigger transforms challenge segments and compares the result to a hidden constant [39]. Unlike sequential triggers, hash-based activation obscures conditions through functional transformation. This nonlinearity enables evaluation of whether increased structural complexity masks observable deviations when concealed conditions are satisfied.

C. Payloads

1) Bias Injection: This payload forces the PUF output to a fixed logic value after activation, introducing deterministic bias into the challenge–response behavior. Activation occurs on the first valid evaluation following trigger assertion, after which all responses become constant and independent of the delay race result. This degrades uniformity and uniqueness while preserving internal switching activity, potentially limiting detectability through structural or power-based analysis. The payload evaluates whether statistical degradation can remain concealed, particularly in nonlinear architectures.

2) Freeze Output: This payload stores the first valid response after activation and replays it for all subsequent evaluations. Internal delay computation continues, but its result is no longer externally propagated. By eliminating challenge dependent variability, freezing output violates the PUF requirement of unique CRP mappings [1]. However, because internal logic continues switching, structural or power based anomalies tend to remain limited. This payload evaluates whether architectural complexity obscures detection of entropy collapse, especially in nonlinear designs where internal activity persists.

IV. Experimental Setup

This section outlines the experimental methodology for evaluating PUF designs.

A. Simulation Flow

All experiments are driven by a unified Verilog testbench. Challenge vectors are pre-generated as 64-bit hexadecimal values and stored in external text files. During simulation, these values are loaded into memory and sequentially applied to one or more parameterized PUF wrapper instances.

Each wrapper selected the PUF architecture, Trojan type, payload, and randomness mode at compile time. The testbench generates a one-cycle start pulse per challenge and logged the resulting response along with activation flags and configuration metadata.

Two operating modes are supported: (1) Standard Mode, full logging of responses and configuration fields for comprehensive understanding, and (2) ML Mode, minimal logging of CRPs to generate datasets for model training.

B. Functional Metric Analysis

Raw simulation logs are exported as CSV files. Metric computation (uniqueness, reliability, uniformity, randomness) is performed using Python scripts that parse the logged data and compute the necessary statistics according to the defined formulas. Reliability experiments

utilize two independent simulation runs with identical configuration parameters.

C. Synthesis Analysis

Each clean and Trojan-infected configuration is synthesized in Vivado. A separate custom synthesis project is used for convenience while maintaining identical implementations, settings, and device parameters. Test cases are selected through a wrapper module, allowing each PUF and Trojan combination to be compiled independently before using the standard Vivado synthesis tool. Resource utilization values are extracted directly from the Vivado synthesis reports, while power estimates are obtained using the Vivado power analysis tool after synthesis. All values are collected from these reports, ensuring that differences across configurations are caused only by the inserted Trojan logic.

D. ML Modeling Attack

For modeling analysis, 50,000 CRPs are generated per configuration using RTL simulation, specifically in ML mode within the custom testbench. Exported datasets are processed in Python, transformed into the Φ feature space for modeling, and evaluated using logistic regression. Training and testing splits are performed using fixed random seeds to ensure consistency and reproducibility across experiments. All data is collected from implementations operating at the RTL; therefore, responses reflect purely structural logic behavior rather than silicon delay asymmetries. The modeling analysis focuses on Trojan-induced functional deviations and post-trigger learnability, rather than the physical modeling resistance of the PUFs themselves.

E. Automation and Reproducibility

All configuration parameters are defined at compile time in the wrapper and testbench. Logging, dataset generation, and file handling are automated to ensure consistency across experimental runs.

V. Experimental Results

This section evaluates the functional, structural, and security characteristics of delay-based PUF architectures under both clean and Trojan-infected conditions. The source code is available on GitHub [1].

A. Functional Analysis

1) Uniqueness: Table I reports uniqueness values across all configurations. Prior to activation, all architectures remain near the ideal 50% target. Clean and dormant Trojan designs are statistically indistinguishable, confirming that embedded trigger logic does not affect inter-instance variation under normal operation. Architectural complexity does not materially influence baseline uniqueness. After Trojan activation, uniqueness degrades sharply across all architectures. Triggered payload behavior increases response correlation, reducing inter-instance differentiation. This effect is consistent across simple and composite designs, indicating that structural complexity does not prevent post-activation degradation.

TABLE I
Uniqueness Values Across PUF Implementations (%)

Configuration	SA-PUF	FA-PUF	XA-PUF	OA-PUF
Clean v. Clean	48.9	49.6	50.2	49.8
Clean v. Sequential Trojan (Dormant)	48.9	49.6	50.2	49.9
Clean v. Sequential Trojan (Triggered)	1.2	0.9	0.6	0.4
Clean v. Hash Trojan (Dormant)	48.9	49.6	50.2	49.9
Clean v. Hash Trojan (Triggered)	1.1	0.8	0.5	0.3
Clean v. Counter Trojan (Dormant)	48.9	49.6	50.2	49.9
Clean v. Counter Trojan (Triggered)	1.3	1.0	0.7	0.5

[1] https://github.com/cars-lab-repo/TroPUF

2) Reliability: Table II reports reliability results across all configurations. Reliability is evaluated by executing two independent simulation runs using identical challenge sets and configurations, and comparing the responses produced in each run. Results report the reliability between the separate runs. Under clean and dormant conditions, all architectures exhibit perfect reliability, with responses remaining consistent across repeated evaluations. Dormant Trojan logic does not disrupt response stability. Following activation, reliability drops to approximately 50% across all architectures. Triggered payload behavior interferes with deterministic response generation, producing instability under identical challenges. This degradation occurs uniformly across architectures, indicating that structural strengthening does not mitigate reliability loss once malicious logic is active.

TABLE II
Reliability Values Across PUF Implementations (%)

Configuration	SA-PUF	FA-PUF	XA-PUF	OA-PUF
Clean v. Clean	100	100	100	100
Clean v. Sequential Trojan (Dormant)	100	100	100	100
Clean v. Sequential Trojan (Triggered)	49.8	50.2	49.5	50.1
Clean v. Hash Trojan (Dormant)	100	100	100	100
Clean v. Hash Trojan (Triggered)	50.3	49.7	50.0	49.6
Clean v. Counter Trojan (Dormant)	100	100	100	100
Clean v. Counter Trojan (Triggered)	49.9	50.4	49.6	50.2

3) Uniformity: Table III reports uniformity values across all configurations. Clean and dormant designs maintain response distributions near the ideal 50% balance, with no measurable bias introduced prior to activation. Architectural variation does not significantly affect uniformity. After activation, uniformity shifts toward extreme bias across all architectures. Triggered payload behavior constrains output distributions, driving responses away from the ideal state. This effect is consistent regardless of structural complexity.

TABLE III
Uniformity Values Across PUF Implementations (%)

Configuration	SA-PUF	FA-PUF	XA-PUF	OA-PUF
Clean	48.7	49.4	50.1	49.8
Sequential Trojan (Dormant)	48.7	49.4	50.1	49.8
Sequential Trojan (Triggered)	99.2	99.5	99.7	99.9
Hash Trojan (Dormant)	48.7	49.4	50.1	49.8
Hash Trojan (Triggered)	99.1	99.6	99.8	99.9
Counter Trojan (Dormant)	48.7	49.4	50.1	49.8
Counter Trojan (Triggered)	99.3	99.4	99.7	99.8

4) Randomness: Table IV reports randomness (entropy) values across all configurations. Under clean and dormant conditions, all architectures exhibit near-maximum entropy, indicating statistically unpredictable behavior. No difference is observed between clean and Trojan-infected designs prior to activation. Following activation, entropy drops sharply across all architectures. Triggered payload behavior reduces response variability and increases predictability. This degradation occurs uniformly across designs, demonstrating that architectural obfuscation does not preserve unpredictability once malicious logic becomes active.

TABLE IV
Randomness Values Across PUF Implementations (%)

Configuration	SA-PUF	FA-PUF	XA-PUF	OA-PUF
Clean	99.89	99.99	99.98	99.99
Sequential Trojan (Dormant)	99.89	99.99	99.98	99.99
Sequential Trojan (Triggered)	2.56	1.98	0.87	0.15
Hash Trojan (Dormant)	99.99	99.99	99.98	99.99
Hash Trojan (Triggered)	1.47	0.92	0.41	0.12
Counter Trojan (Dormant)	99.99	99.99	99.98	99.99
Counter Trojan (Triggered)	1.43	1.01	0.76	0.20

B. Synthesis Analysis

1) Lookup Table Utilization: Fig. 6 reports LUT utilization across all configurations. Trojan insertion introduces additional combinational logic due to trigger and control circuitry. However, the increases remain slight and consistent across Trojan types. Relative overhead varies by architecture. Simpler designs exhibit greater sensitivity, while more complex architectures absorb the additional logic with reduced relative impact. Overall, LUT-based analysis alone provides limited discriminatory power for detecting dormant Trojans, particularly in designs with higher inherent complexity.

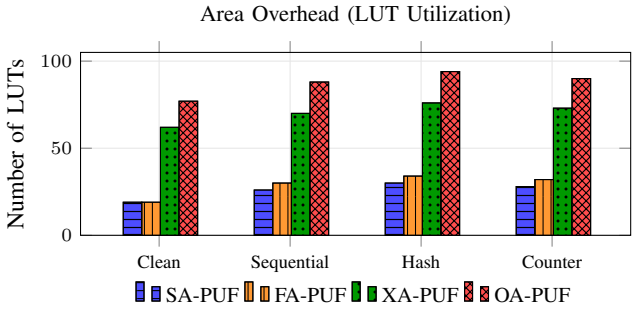

Fig. 6. Lookup Table (LUT) utilization across PUF architectures.

2) Flip-Flop Utilization: Fig. 7 reports flip-flop utilization across all configurations. Register usage increases primarily, however slightly, in stateful trigger implementations, such as counter mechanisms, while stateless triggers introduce smaller overhead. The relative impact depends on baseline sequential complexity. Architectures with minimal native register usage exhibit more visible proportional increases, whereas more complex designs absorb the additional registers with reduced relative effect. Consequently, register based analysis provides limited effectiveness for identifying dormant Trojans.

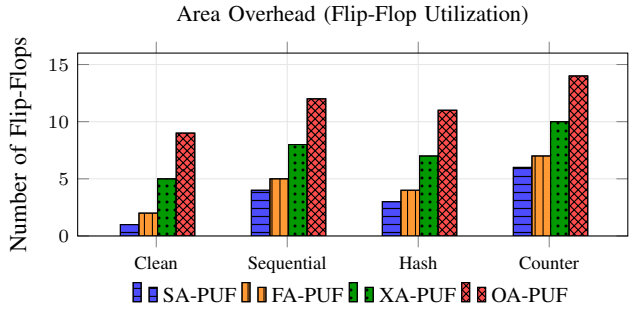

Fig. 7. Flip-Flop (FF) utilization across PUF architectures.

3) Power Consumption: Fig. 8 reports total on-chip power estimates across configurations. Under dormant conditions, power values remain within expected estimation noise, with no consistent or monotonic relationship observed across Trojan types. Dormant Trojan logic does not produce a distinguishable power signature. Any incremental consumption is masked by the surrounding switching activity of the PUF circuitry. Architectures with higher inherent activity further reduce observable impact, limiting the effectiveness of power-based detection.

C. ML Modeling Attack

Table V reports logistic regression prediction accuracy across implementations of all evaluated PUF architectures. All clean and dormant configurations showcase prediction accuracy around 50%,

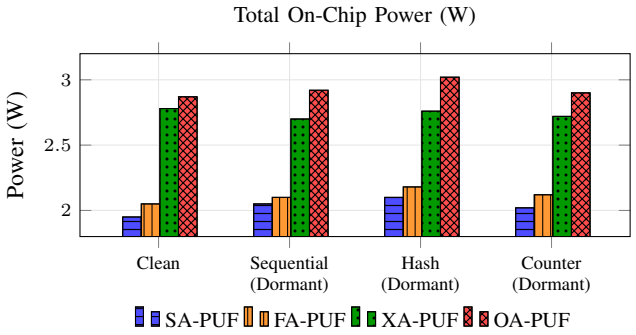

Fig. 8. Total on-chip power across PUF architectures.

corresponding to near random guessing. This behavior is expected considering experiments are conducted at the RTL level. As a result, the PUF responses are statistically balanced and contain no structural bias that a learning algorithm can exploit.

Dormant Trojan insertion does not alter modeling accuracy across any architecture. Clean and infected (pre-trigger) implementations remain indistinguishable. This indicates that inactive Trojan logic does not introduce observable artifacts at the structural level that can be learned by a model.

In contrast, once Trojans are triggered, prediction accuracy increases dramatically to above 99%. Trojan activation exploits entropic PUF behavior, making responses predictable. This transition confirms that Trojan presence fundamentally alters the functional characteristics of the PUFs only after activation, while remaining completely stealthy prior to triggering, proving that while dormant, they can not be detected via modeling.

TABLE V
ML MODEL PREDICTION ACCURACY ACROSS PUF IMPLEMENTATIONS (%)

Configuration	SA-PUF	FA-PUF	XA-PUF	OA-PUF
Clean	49.3	51.3	49.6	49.7
Sequential Trojan (Dormant)	49.3	51.3	49.6	49.7
Hash Trojan (Dormant)	49.3	51.3	49.6	49.7
Counter Trojan (Dormant)	49.3	51.3	49.6	49.7
Sequential Trojan (Triggered)	99.5	99.7	99.8	99.7
Hash Trojan (Triggered)	99.7	99.7	99.8	99.9
Counter Trojan (Triggered)	99.4	99.6	99.7	99.6

VI. CONCLUSIONS

In this work, we showed that hardware Trojans could be successfully embedded within PUF architectures while remaining undetectable. Dormant Trojan-infected designs exhibited no meaningful behavioral differences compared to clean instances, but once activated, Trojans caused immediate changes in PUF behavior.

This reveals a fundamental limitation in current PUF safety and validation methodologies. In short, satisfying the regular criteria does not imply trustworthiness. The same characteristics that make PUFs effective security primitives can also enable them to conceal malicious logic, thereby undermining the trust they are intended to provide. A PUF can appear statistically ideal, yet still contain malicious insertions. As a result, existing Trojan detection approaches and PUF validation pipelines are insufficient when applied in isolation. New detection strategies must move beyond current assumptions, and address the presence of malicious logic embedded directly within the security primitive itself.

References

[1] Frederik Armknecht, Daisuke Moriyama, Ahmad-Reza Sadeghi, and Moti Yung. Towards a unified security model for physically unclonable functions. In *Cryptographers' Track at the RSA Conference*, pages 271–287, 2016.

[2] SV Sandeep Avvaru and Keshab K Parhi. Feed-forward xor pufs: Reliability and attack-resistance analysis. In *Proceedings of the 2019 Great Lakes Symposium on VLSI*, pages 287–290, 2019.

[3] Georg T Becker. The gap between promise and reality: On the insecurity of xor arbiter pufs. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 535–555, 2015.

[4] Shivam Bhasin and Francesco Regazzoni. A survey on hardware trojan detection techniques. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2021–2024. IEEE, 2015.

[5] Wenjie Che, Fareena Saqib, and Jim Plusquellic. Puf-based authentication. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 337–344, 2015.

[6] Qingqing Chen, György Csaba, Paolo Lugli, Ulf Schlichtmann, and Ulrich Rührmair. The bistable ring puf: A new architecture for strong physical unclonable functions. In *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*, pages 134–141, 2011.

[7] Michael Dominguez and Amin Rezaei. Cycpuf: cyclic physical unclonable function. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, 2024.

[8] Mohammad Ebrahimabadi, Mohamed Younis, Wassila Lalouani, and Naghmeh Karimi. A novel modeling-attack resilient arbiter-puf design. In *2021 34th International Conference on VLSI Design and 2021 20th International Conference on Embedded Systems (VLSID)*, pages 123–128, 2021.

[9] Fatemeh Ganji, Domenic Forte, and Jean-Pierre Seifert. Pufmeter a property testing tool for assessing the robustness of physically unclonable functions to machine learning attacks. *IEEE Access*, 7:122513–122521, 2019.

[10] Yansong Gao, Said F Al-Sarawi, and Derek Abbott. Physical unclonable functions. *Nature Electronics*, 3(2):81–91, 2020.

[11] Blaise Gassend, Dwaine Clarke, Marten Van Dijk, and Srinivas Devadas. Silicon physical random functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 148–160, 2002.

[12] Blaise Gassend, Daihyun Lim, Dwaine Clarke, Marten Van Dijk, and Srinivas Devadas. Identification and authentication of integrated circuits. *Concurrency and Computation: Practice and Experience*, 16(11):1077–1098, 2004.

[13] Kevin Immanuel Gubbi, Banafsheh Saber Latibari, Anirudh Srikanth, Tyler Sheaves, Sayed Arash Beheshti-Shirazi, Sai Manoj PD, Satareh Rafatirad, Avesta Sasan, Houman Homayoun, and Soheil Salehi. Hardware trojan detection using machine learning: A tutorial. *ACM Transactions on Embedded Computing Systems*, 22(3):1–26, 2023.

[14] S Hemavathy and VS Kanchana Bhaaskaran. Arbiter puf—a review of design, composition, and security aspects. *IEEE Access*, 11:33979–34004, 2023.

[15] Yier Jin, Nathan Kupp, and Yiorgos Makris. Experiences in hardware trojan design and implementation. In *2009 IEEE international workshop on hardware-oriented security and trust*, pages 50–57, 2009.

[16] Ramesh Karri, Jeyavijayan Rajendran, and Kurt Rosenfeld. Trojan taxonomy. In *Introduction to hardware security and trust*, pages 325–338. Springer, 2011.

[17] Sandeep S Kumar, Jorge Guajardo, Roel Maes, Geert-Jan Schrijen, and Pim Tuyls. The butterfly puf protecting ip on every fpga. In *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, pages 67–70, 2008.

[18] Huafeng Liu, Hongwei Luo, and Liwei Wang. Design of hardware trojan horse based on counter. In *2011 International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering*, pages 1007–1009, 2011.

[19] Y. Lyu and P. Mishra. Scalable activation of rare triggers in hardware trojans by repeated maximal clique sampling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(7):1–14, 2021.

[20] Abhranil Maiti, Vikash Gunreddy, and Patrick Schaumont. A systematic method to evaluate and compare the performance of physical unclonable functions. In *Embedded systems design with FPGAs*, pages 245–267. Springer, 2012.

[21] Jordan Maynard and Amin Rezaei. Reconfigurable run-time hardware trojan mitigation for logic-locked circuits. In *Proceedings of the IEEE 17th Dallas Circuits and Systems Conference (DCAS)*, 2024.

[22] Phuong Ha Nguyen, Durga Prasad Sahoo, Chenglu Jin, Kaleel Mahmood, Ulrich Rührmair, and Marten Van Dijk. The interpose puf: Secure puf design against state-of-the-art machine learning attacks. *Cryptology ePrint Archive*, 2018.

[23] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber. Modeling attacks on physical unclonable functions. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 237–249, 2010.

[24] Ulrich Rührmair, Jan Sölter, and Frank Sehnke. On the foundations of physical unclonable functions. *Cryptology ePrint Archive*, 2009.

[25] Sauvagya Ranjan Sahoo, Sudeendra Kumar, and Kamalakanta Mahapatra. A novel ropuf for hardware security. In *2015 19th international symposium on VLSI design and test*, pages 1–2, 2015.

[26] Md Abu Bokor Siddik and Sk Hasibul Alam. Puf-based hardware trojan: design and novel attack on encryption circuit. In *2023 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, pages 1–5, 2023.

[27] Shuqin Su, Min Zhu, Hanning Wang, Bohan Yang, and Leibo Liu. A survey on the security of pufs. In *Journal of Physics: Conference Series*, volume 1993, page 012031, 2021.

[28] G Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th annual design automation conference*, pages 9–14, 2007.

[29] Mohammad Tehranipoor and Farinaz Koushanfar. A survey of hardware trojan taxonomy and detection. *IEEE design & test of computers*, 27(1):10–25, 2010.

[30] Shubham Kumar Tiwari and SR Ramesh. An efficient approach for hardware trojan detection based on side-channel analysis. In *2023 IEEE 20th India Council International Conference (INDICON)*, pages 85–90, 2023.

[31] Johannes Tobisch and Georg T Becker. On the scaling of machine learning attacks on pufs with application to noise bifurcation. In *International Workshop on Radio Frequency Identification: Security and Privacy Issues*, pages 17–31, 2015.

[32] Arunkumar Vijayakumar, Vinay C Patil, Charles B Prado, and Sandip Kundu. Machine learning resistant strong puf: Possible or a pipe dream? In *2016 IEEE international symposium on hardware oriented security and trust (HOST)*, pages 19–24, 2016.

[33] Rahul Vishwakarma and Amin Rezaei. Risk-aware and explainable framework for ensuring guaranteed coverage in evolving hardware trojan detection. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023.

[34] Rahul Vishwakarma and Amin Rezaei. Uncertainty-aware hardware trojan detection using multimodal deep learning. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024.

[35] Xinmu Wang, Seetharam Narasimhan, Aswin Krishna, Tatini Mal-Sarkar, and Swarup Bhunia. Sequential hardware trojan: Side-channel aware design and placement. In *2011 IEEE 29th International Conference on Computer Design (ICCD)*, pages 297–300, 2011.

[36] Yale Wang, Chenghua Wang, Chongyan Gu, Yijun Cui, Maire O'Neill, and Weiqiang Liu. Theoretical analysis of delay-based pufs and design strategies for improvement. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2019.

[37] Nils Wisiol, Georg T Becker, Marian Margraf, Tudor AA Soroceanu, Johannes Tobisch, and Benjamin Zengin. Breaking the lightweight secure puf: Understanding the relation of input transformations and machine learning resistance. In *International Conference on Smart Card Research and Advanced Applications*, pages 40–54, 2019.

[38] Nils Wisiol, Christopher Mühl, Niklas Pirnay, Phuong Ha Nguyen, Marian Margraf, Jean-Pierre Seifert, Marten Van Dijk, and Ulrich Rührmair. Splitting the interpose puf: A novel modeling attack strategy. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 97–120, 2020.

[39] 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.

[40] Chen Zhou, Keshab K Parhi, and Chris H Kim. Secure and reliable xor arbiter puf design: An experimental study based on 1 trillion challenge response pair measurements. In *Proceedings of the 54th Annual Design Automation Conference 2017*, pages 1–6, 2017.