DSMLock: Low Overhead Logic Locking for System Security with Design Space Modeling

Robi Paul, *Student Member, IEEE*, Long Lam, *Student Member, IEEE*, Maksym Melnyk, *Student Member, IEEE* and Michael Zuzak, *Member, IEEE*

***Abstract*—Integrated circuits (ICs) are increasingly manufactured in untrusted facilities, raising concerns regarding intellectual property (IP) protection. Logic locking has emerged as a solution to address these concerns by corrupting a circuit's functionality unless a correct secret key is applied. However, prior work has shown that system-level phenomena can undermine module-level locking strategies, motivating the need for system-aware logic locking configuration. In this paper, we propose DSMLock, a design space modeling algorithm that identifies favorable logic locking configurations for arbitrary ICs by simulating a small, carefully selected subset of the design space to generate system-level predictive models. DSMLock efficiently identifies promising solutions that meet arbitrary, designer-specified security goals with minimal power/area overhead. To evaluate the proposed algorithm, we perform two experiments. First, an exhaustive simulation of the considered logic locking design space for a RISC-V ALU demonstrated that our models achieved an average $R^2 > 0.99$ across all design objectives and identified a locking configuration within 96% of the global optimum after simulating less than 3.6% of the design space. Second, when comparing DSMLock against three module-level locking approaches for a RISC-V processor and an APRSC communication interface, we evaluated two cost functions: power-focused and area-focused. Under the power-focused cost function, DSMLock achieved average power reductions of 42.4% and 6.0% for the locked modules in the RISC-V and APRSC designs, respectively. Under the area-focused cost function, DSMLock achieved average area reductions of 6.1% for the RISC-V locked modules and 3.3% for the APRSC locked modules. Moreover, of the 3 evaluated locking approaches, DSMLock was the only approach to meet all specified design objectives across both design objectives for all test cases.**

***Index Terms*—DSMLock, Logic Locking, Design Space Exploration (DSE), Design Space Modeling (DSM)**

I. Introduction

INCREASING costs and complexity for integrated circuit (IC) manufacturing, particularly at advanced process nodes, coupled with pressure to reduce time-to-market, has led to the widespread adoption of untrusted facilities for manufacturing and testing. However, such an approach raises significant concerns regarding IP rights violations by these untrusted facilities, posing substantial financial risks [1]. Such supply chain security challenges are referred to as the *untrusted foundry problem* [2].

Logic locking was developed to mitigate security concerns related to the untrusted foundry problem by locking the true functionality of modules containing high-value IP behind a secret key [3]. An incorrect key results in errors that alter the module's functionality, thereby preventing unauthorized use. A sizable amount of work on logic locking explores these schemes through a module-level lens, relying on the assumption that errant functionality in a module is sufficient to protect unauthorized IP use [4]. However, recent studies have shown that such assumptions are often limited in practice [5]–[7]. For example, system-level phenomena, such as error resilience and input space utilization, may diminish the security provided by logic locking implemented at the module level when an IC is considered as a whole [5]. This prompted a shift towards architecture and system-level logic locking implementations, including gate-level constructions with architectural evaluations and HLS strategies [8], [9]. Despite such high-level locking approaches, most research relies on the assumption that a designer has a pre-configured locking scheme for the design and must only consider optimizing security and attack resilience. This is a non-trivial assumption that substantially impacts design goals (e.g., power, area, etc.) [4].

In this work, we aim to address this problem, exploring how to best configure system-level logic locking in an IC to meet system objectives. Our work focuses on the following key design challenge: *given a list of modules containing critical IP that must be protected and an arbitrary set of design objectives (e.g., power and area budget, attack time, etc.), identify a set of locking techniques, their corresponding key length, and the modules they are implemented in to optimize for designer-specified system design goals*. Unfortunately, this design space is not intuitive due to the many variables involved and the complex interaction between these variables. Moreover, exhaustive simulation or ad-hoc approaches to explore this design space are infeasible. To overcome this, we propose design space modeling (DSM) to construct mathematical models of the locking design space from a small number of simulated locking configurations. DSM has been successfully applied to varied design space exploration (DSE) problems with complex, multivariate design spaces [10], [11]. This leads to our primary goal: **to develop a DSM algorithm to model the system-level design space for logic locking and employ these models to identify favorable logic locking configurations in arbitrary ICs optimizing for designer-specified objectives.**

A. Contributions

We propose DSMLock, a novel algorithm that integrates DSE and mathematical modeling to optimize logic locking configurations while balancing multiple designer-specified objectives such as area, power, and security. DSMLock employs

This work is supported by the National Science Foundation (NSF) under Grant 2245573.

R. Paul, L. Lam, M. Melnyk, and M. Zuzak are with the Department of Computer Engineering at Rochester Institute of Technology, Rochester, NY, USA. Email: {rp7248, ll5530, mm6878, mjzeec}@rit.edu

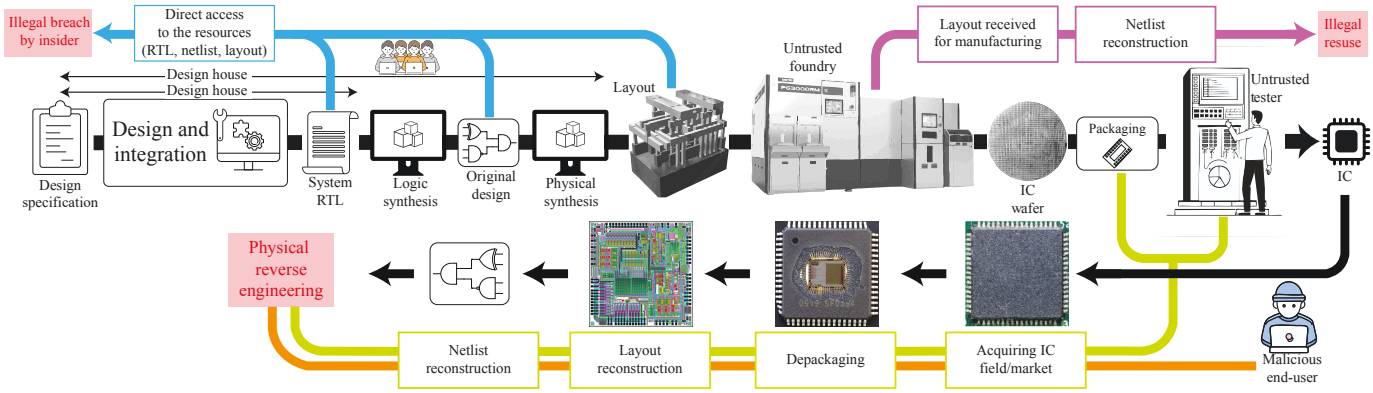

Fig. 1. Potential threats at different stages of IC design and manufacturing life cycle. This figure is based on a figure from [12].

DSM to construct mathematical models to quantify the cost and performance of different locking schemes. These models are then used in a numerical optimization framework to identify configurations that meet specific, designer-defined objectives in a target system. For example, designers can specify an objective such as minimizing area overhead subject to a power consumption constraint, or they may set a security target while allowing for minor trade-offs in power and area. Our key contributions to this work include:

- We introduce a unified system-level security metric that quantifies the underlying attack resilience and error rate for an unauthorized user of a logic-locked IP.
- We propose a guided DSM algorithm to build mathematical models that relate locking configuration (e.g., key length, locking technique) to design objectives (e.g., power, area, security).
- We explore multiple initialization strategies to produce DSMLock's initial design space models. This allows designers to tailor the modeling process and balance computational cost with the solution quality.
- We propose a numerical optimization algorithm, based on generalized simulated annealing (GenSA), to identify promising logic locking configurations based on designer goals. The proposed algorithm searches the entire design space, which is exponentially larger than prior work that used greedy assumptions to limit complexity [13].
- We make open-source the DSMLock framework that builds design space models and identifies favorable locking configurations for arbitrary cost functions [14].

DSMLock assumes the locking technique and the corresponding gate-insertion/configuration algorithm are preselected. DSMLock's role is to build predictive models and identify favorable locking configurations. To evaluate the fit and predictive capability of the models generated by DSMLock, we performed a small, exhaustive DSE simulation of logic locking configurations in the ALU in a RISC-V processor [15]. Our results indicate that the models produced by DSMLock are highly predictive, modeling the power, area, security, and SAT attack runtime of a locking configuration with an average $R^2 > 0.99$. Moreover, DSMLock identified promising configurations that achieved 96% of the globally optimal solution while simulating only 3.6% of the design space. Further evaluations of DSMLock on a complete RISC-V processor [15] and an APRSC communication interface [16] using power and area focused cost functions. Compared with conventional locking strategies [17]–[19], DSMLock's locking configurations reduced average power by 42.4% (RISC-V) and 6.0% (APRSC) and average area by 6.1% (RISC-V) and 3.3% (APRSC). Additionally, of the 3 evaluated locking strategies, DSMLock was the only method that satisfied all design constraints (area, attack time, security, and power) for all considered scenarios.

II. Preliminaries

High costs and increased complexity in modern IC design have forced the semiconductor industry to increasingly rely on third-party entities, a practice that introduces potential IP violations and financial risks. As shown in Fig. 1, such risks can arise from various sources: rogue insiders at the design house who gain unauthorized access to design assets (e.g., RTL, netlist, layout), untrusted factories that reconstruct netlists from manufacturing layouts for illicit reuse, and malicious end-users who perform physical reverse engineering (e.g., depackaging and layout reconstruction). While many threats exist in the IC supply chain, this work focuses specifically on the *untrusted foundry problem* [4]. This threat is particularly

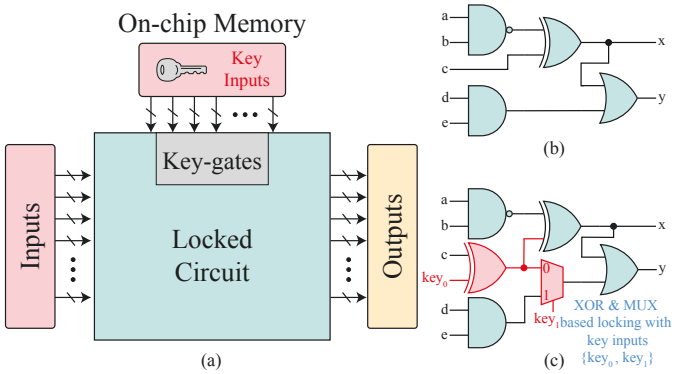

Fig. 2. (a) Overview of a logic-locked design flow, (b) Original unlocked circuit, and (c) Example locked circuit using XOR and multiplexer elements.

challenging as designers have to disclose their designs to external manufacturing parties through file formats like GDSII, which can be reverse-engineered into a complete netlist [2].

A. Logic Locking

Logic locking addresses the *untrusted foundry problem* by rendering IC functionality dependent on a locking key. A block diagram for logic locking, along with a small example circuit locked using a gate-level locking technique, is shown in Fig. 2. This is achieved by adding extra primary inputs, known as key inputs, which drive combinational logic within a target module. If an incorrect key is applied, the module produces erroneous outputs, derailing the target module's intended functionality and preventing unauthorized use [4]. The effectiveness of logic locking schemes can be evaluated through two key metrics: error severity and attack resilience [20]. Error severity refers to how well logic locking disrupts a device's function, typically measured by the fraction of inputs that produce incorrect outputs for an invalid key. Attack resilience is the ability of the locking scheme to resist an attack. A higher error severity typically means unauthorized users will have more difficulty using unauthorized IP effectively, while stronger attack resilience indicates the scheme is less susceptible to automated attack methods such as SAT-based attacks [21], [22]. Several state-of-the-art locking schemes include stripped functionality logic locking (SFLL) [17], CASLock [18], and others [3], [23]–[27]. While these schemes offer important advances, they often face limitations when viewed as part of a broader system, as discussed in Sec. III.

B. Design Space Exploration and Modeling

DSE systematically explores and identifies favorable design configurations to meet designer-specific objectives, such as performance, power, area, or security, from a large design space [13], [28]–[31]. However, exhaustively exploring a vast design space is often impractical due to computational constraints. DSM aims to address this challenge by constructing mathematical models to estimate design objectives, enabling efficient identification of promising design configurations [10], [11], [13]. DSE and DSM have been widely adopted in architectural design [13], [29], 3D IC [11], [32], and other domains [30] to identify favorable design configurations from a large design space.

DSM relies on statistical modeling to capture the complex and non-intuitive relationships between design parameters and system objectives [33], [34]. The complex interactions between design objectives prevent analytical derivations, requiring a data-driven approach for effective DSE [31], [35], [36]. In this work, we employ several statistical modeling techniques, including regression and analysis of variance (ANOVA) to generate design space models for the logic-locking configuration design space (see Sec. VI-C3).

TABLE I
EXAMPLE SYSTEM-LEVEL DESIGN SPACE FOR LOGIC LOCKING.

	RISC-V	**APRSC**
	Possible variable configurations	
Locked module:	ALU, Decoder, Branch predictor	Adder, Limiter module, Floating-point module, Upper bound module, Accumulator, Floating-point multiplier
Locking technique:	{SFLL [17], CASLock [18], SLL [19]}	
Key length (Bits):	{0, 16, 32, 48, 64}	{3, 6, 8, 16, 32, 48, 64, 128, 256}

C. Related Work

Prior work has explored high-level approaches (i.e., system or architecture-level) to mitigate the untrusted foundry problem. These include secure scan-chains [37] and behavioral locking techniques applied during high-level synthesis or at the register transfer level [29], [38]. While these approaches consider high-level security, they do not use logic locking. This overlooks the body of work developing provably secure and low-overhead logic locking. For example, the behavioral locking used by TAO [39] is vulnerable to SAT-based attacks, whereas modern logic locking techniques [17], [18], [20] are resilient against such attacks. Several studies have explored the role of logic locking in system security [8], [9], [40]. However, these approaches generally assume that designers have already selected and implemented specific locking techniques and their primary focus is on fine-tuning locking configurations within a module to optimize security goals. In contrast, in this work, we consider a broader problem where a designer has only specified modules containing critical IP to be locked and the design constraints/goals to be satisfied with the locking solution, rather than predefining specific locking techniques or specific configurations as is done in previous work [5], [8], [9], [17], [18], [20], [40].

D. Threat Model

We consider an untrusted foundry adversary who aims to use the IC in an unauthorized manner. They can take any strategy using:

1) A locked IC netlist from GDSII reverse-engineering [2].
2) An activated, black-box oracle IC from testing facilities or the open market that can be queried with arbitrary inputs.

A defense must 1) resist attacks against locking (e.g., SAT attack [21]); and 2) inject enough error to stop unauthorized IP use for a wrong key. This model is consistent with prior work [17], [18], [20], [41]. We note that DSMLock is threat-model-agnostic. This oracle-equipped attacker in selected only for the evaluation in Sec. VI. Alternative threat models (e.g., oracle-less) can be considered by selecting suitable attacks for model generation.

III. MOTIVATION AND PROBLEM FORMULATION

DSE aims to identify a favorable configuration of m decision variables (degrees of freedom) that optimizes n objective

values [34]. DSM addresses this DSE problem by introducing a fitness function, $f(m)$, which maps each m-dimensional variable configuration to an n-dimensional objective vector in the solution space (i.e., a prediction for the n objective values). This fitness function can be used with optimization strategies (e.g., GenSA) to identify promising candidate solutions that optimize for arbitrary, designer-specified objectives.

In this work, we apply DSM to address the system-level logic locking configuration problem. Specifically, we formalize the design space as an m-variable system, where each of the m variables corresponds to a particular locking choice in a "candidate module" (i.e., a critical IP block). Because each module can use different locking techniques with varying key lengths, the total number of possible configurations grows exponentially. For instance, Table I shows a RISC-V processor with three critical modules, each supporting three locking schemes and key lengths of $|k| = \{0, 16, 32, 48, 64\}$. Even this relatively small example yields $(5^3)^3 = 1,953,125$ possible configurations. Exhaustively simulating each possibility is therefore computationally infeasible, underscoring the need for a systematic exploration framework.

Based on the m-variable locking configuration, we explore the use of DSM to quantify a set of n objective values. In this work, each m-variable locking configuration is evaluated according to four critical objectives: (1) system security (see Sec. IV-A), (2) SAT attack runtime, (3) power overhead, and (4) area overhead[1]. These objective values are selected because they are commonly used to assess logic locking techniques in prior work [17]–[19]. However, they are selected without the loss of generality as any set of objective values could be selected by the designer based on their design goals. Finally, an arbitrary cost function, specified by the designer, must be defined to aggregate the objective values into quantifiable design goals (e.g., minimizing power subject to an area and security constraint).

IV. System Security Metric Formulation

Principled system design requires quantifiable design objectives. As such, an effective DSM approach requires a quantifiable metric to systematically assess different locking configurations for their security. However, prior studies often rely on either qualitative assessments [8], [41] (for example, "restrict unauthorized use") or module-level metrics [18] that do not capture the system-wide security implications of a locking scheme. To address this limitation, we introduce the S-Metric, a unified measure that accounts for both the frequency and severity of locking-induced errors. Capturing both how often errors (i.e., error frequency) occur and how damaging they are (i.e., error sensitivity) ensures a more holistic understanding of how well the locking scheme deters unauthorized use, whether through many small disruptions or fewer but more harmful ones. The S-Metric is formally defined in Eq. 1, providing a quantitative assessment of security effectiveness at the system level.

[1]While timing (i.e., performance) could serve as an additional metric, we assume the designer is unwilling to degrade clock frequency for locking. Hence, all designs are subject to a fixed clock constraint and timing is excluded as an objective.

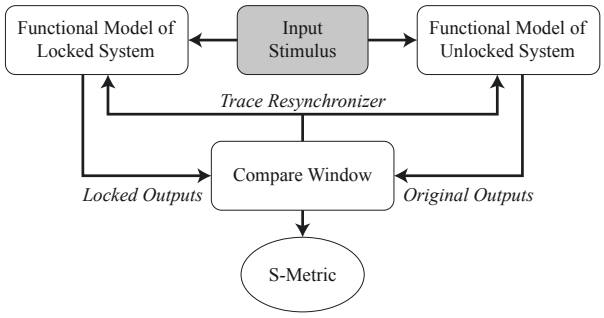

Fig. 3. Method to quantify proposed security metric (S).

$$S - Metric = \frac{inputs\ per\ observed\ error\ (I_e)}{inputs\ until\ recovery\ (I_r)} \quad (1)$$

The *inputs per observed error* (I_e) metric is the average number of input patterns required to trigger an observable output deviation on average. This value indicates how frequently errors occur under incorrect input keys and is measured at the system output during simulation. By capturing outputs at a system level, our approach naturally includes the effects of error resilience mechanisms such as masking and redundancy, which can delay or obscure errors. This differs from module-level locking implementations, where outputs are collected at the module output, nullifying any possibility of error masking. At the system level, the effects of error resilience become important as an error introduced in one module might later be masked or mitigated by downstream components. The *inputs until recovery* (I_r) metric represents the number of subsequent input cycles required for the system to restore correct functionality after an erroneous output, reflecting the severity of locking-induced errors while incorporating factors like module importance. Together, these metrics define the S-Metric, where **a lower value implies stronger security, as it indicates more frequent and persistent errors that increase the difficulty of unauthorized operation.** Note that this approach is similar to the system-level security metrics proposed in [5]. However, in this work, we produce a single, unified security metric for locking that can be optimized.

A. Measuring the System Security Metric (S)

The proposed S-Metric depends on the specific inputs applied to the IC, the chosen locking configuration, and the characteristics of the locked system. Consequently, the proposed S-Metric is unique per design and requires to be evaluated on a per-design basis. To achieve this, we propose a measurement framework in which two functional simulations of the IC run in parallel: one with the correct key and one with a random incorrect key (see Fig. 3). Both simulations receive the same sequence of inputs, and their outputs are compared within a designated compare window each cycle. Any discrepancy indicates that locking-induced errors have caused the locked and unlocked designs to diverge functionally, which constitutes a critical error.

Once a divergence is detected, the framework tracks how many input cycles elapse before the locked system's outputs

once again match those of the unlocked system. The time to realign is recorded as the *inputs-until-recovery* (I_r), reflecting the severity of the error. Meanwhile, the number of input cycles required to first observe each error is recorded as the *inputs per observed error* (I_e), indicating how frequently discrepancies arise. With enough cycles, when circuits exhibit identical output indicating recovery, the two output traces are re-synchronized to reset the compare window, and the simulation continues, effectively resolving the error. By repeating this process, we can measure the system-wide S-Metric for arbitrary locking configurations across a broad range of ICs.

V. Overview of DSMLock Algorithm

The proposed DSMLock algorithm efficiently explores the system-level logic locking design space to identify high-quality logic locking configurations capable of optimizing for designer-specified objectives. It builds predictive models from a small sample of simulated configurations to estimate quantifiable design metrics. These models then guide the search toward regions likely to yield high-quality locking solutions.

DSMLock requires two inputs: an m-variable system-level locking design space and a designer-specified cost function reflecting predefined system goals and constraints. In this work, for each lockable module, DSMLock defines two variables: (1) the locking technique used and (2) the key length for that chosen technique. The designer-specified cost function then formalizes the system-level goals and design requirements for an effective logic-locking configuration for the system. For example, a design objective could be to minimize power overhead while satisfying constraints on area utilization, SAT runtime, and the S-Metric. Upon termination, DSMLock returns two outputs: 1) a design space model, a function $f(m)$, which estimates four critical objective values (system-level security metric (S-Metric), SAT runtime, power, and area) for any m-variable locking configuration; and 2) a predicted favorable locking configuration derived from these models based on the designer-specified cost function.

We demonstrated the complete DSMLock algorithm through a block diagram as in Fig. 4. The proposed DSMLock algorithm first collects η initial design space points. Depending on the designer's preference, these points may be chosen either uniformly or at random from the design space (see Sec. V-A). Regression functions are then used to generate a model from this data. We outline the specific regression functions and how they were selected in Sec. V-B. The regression model predicts the objective values for the locking solutions, facilitating comparison between simulation results and model estimates from exhaustive simulation to assess accuracy. The stopping criteria are then evaluated to terminate modeling if model accuracy is above a threshold or if the design space becomes oversampled, risking overfitting (see Sec. V-E). If termination conditions are not met, the algorithm identifies a region of interest (ROI) based on the model and the designer-specified cost function, highlighting promising areas for favorable design configurations (see Sec. V-C). DSMLock then selects the additional design space points from this ROI uniformly or at random according to the designer's initial sampling approach and simulates each candidate locking configuration to measure each design objective (see Sec. VI-A). The model is then re-generated with these added data points, and the process continues until the stopping criteria are met. Upon termination, the algorithm delivers two outputs: the final design space model, $f(m)$, and possible favorable locking configurations. The algorithm has been made open-source [14]. The details of each block in Fig. 4 are discussed for the remainder of this section.

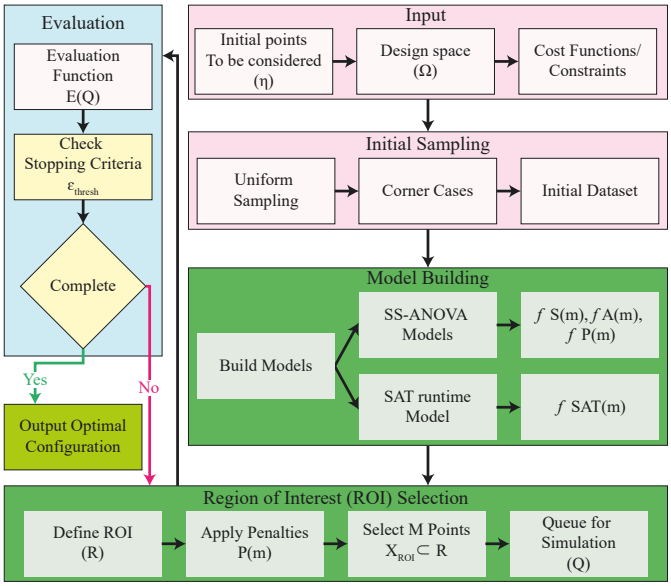

Fig. 4. Overview of proposed overall algorithm.

A. Initial Sampling of Design Space

Creating an effective initial model requires balancing oversampling, which can lead to longer runtime and overfitting, and under-sampling, which may produce a model too poor to guide further simulation. To generate the initial model, we consider two possible strategies: uniform or random sampling of the design space, based on both theoretical considerations and the experimental results presented in Sec. VI-C2. Uniform sampling (i.e., equally spaced points) yields consistent and repeatable results in a single run. However, it may occasionally miss certain promising areas if the sampling grid aligns unfavorably with the design space. Random sampling, on the other hand, can sometimes discover high-quality solutions not covered by a uniform grid, though it has higher variance and may require multiple runs to achieve stable outcomes. Finally, the number of simulated points used to generate initial models, defined as η, is left to the user as design space complexity may vary widely by design, locking scheme, or other application-specific context.

B. Building Models and Regression Functions

A suitable regression function is essential for accurately modeling the design space based on the design objectives (i.e., the cost function). To identify an effective regression

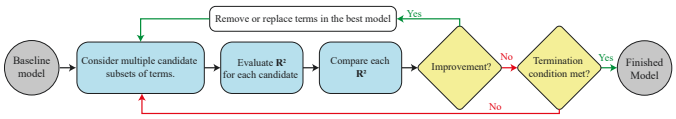

Fig. 5. Progressive term selection algorithm for SSANOVA model generation.

function for each objective, we evaluated multiple candidates (i.e., Smoothing Spline ANOVA (SSANOVA) [42], linear regression, logistic regression, polynomial fitting, and exponential models) by constructing models on small subsets of design space and assessed their predictive accuracy through an exhaustive characterization process (see Sec. VI-C3). Here, we avoid higher-capacity machine learning models (e.g., deep neural networks, decision trees) as their training would require substantially larger labeled datasets. Because each data point corresponds to a comprehensive synthesis and security evaluation of a locked design, which takes hours per instance, such data volumes are impractical to obtain. For each objective, we select the regression function that yields the highest predictive accuracy (i.e., R^2 value), recognizing that different objectives may favor different models.

1) Power, Area, and S-Metric Models: For power, area, and S-Metric objectives, SSANOVA outperformed all other evaluated regression functions in the RISC-V ALU (see Sec. VI-C3), achieving an average $R^2 > 0.99$ after simulating only 3.6% of the design space (see Section VI-C4). Consequently, we have adopted SSANOVA [42] to produce models for these three design objectives. SSANOVA fits smoothing splines (piece-wise cubic functions) to a dataset using the analysis of variance (ANOVA) method [42]. ANOVA is a statistical technique that analyzes the underlying sources of variance in a population and generates a model as a function of descriptive properties. An SSANOVA model, $f(m)$, is represented as a function of independent variables m = $\{m_1, m_2, ..., m_n\}$ as depicted in Eqn. 2 [42]. Each unique subset of these variables is called a term, whose order corresponds to the number of variables in that subset. SSANOVA represents each term as a smoothing spline, $\{f_1, ..., f_{1,2,...,n}\}$. The final model, $f(m)$, is the sum of all smoothing splines.

$$f(\mathbf{m}) = c + \sum_{i=1}^{n} f_i(m_i) + \sum_{i=1}^{n} \sum_{j=i+1}^{n} f_{i,j}(m_i, m_j) + \cdots f_{1,2,\ldots,n}(m_1, m_2, \ldots, m_n) + \epsilon \quad (2)$$

To create system-level models, we propose an iterative, dependency-aware approach in which SSANOVA refines output functions for each design objective. DSMLock begins with a baseline model, that includes only main (first-order) effects, then iteratively evaluates the fit for multiple candidate models at each step (see Fig. 5). Unlike standard stepwise methods, which only add or remove one term at a time [13], this approach evaluates multiple candidate models in each iteration and accepts any change (addition, removal, or replacement of terms) that improves R^2. Each candidate model's adjusted R^2 is computed via a randomly selected set of validation points and adopts the current best model. The process repeats until no further modifications improve in R^2. In the end, SSANOVA outputs a model that captures complex relationships among variables consisting of only first and second order terms.

2) SAT Attack Runtime: Many locking schemes are designed to resist SAT attacks [17], [18], [20], making exhaustive simulations to quantify SAT runtime computationally infeasible for many locking solutions. This limitation restricts the regions of the design space available for empirical modeling. To overcome this challenge, we adopt an alternative approach for runtime modeling that leverages prior theoretical derivations of SAT attack complexity [20], [43]. We use these theoretical results to guide our selection of a regression function for each locking scheme. For example, the Anti-SAT locking scheme [44] demonstrated that the number of SAT attack iterations and, by extension, the runtime grows exponentially with key length (in bits). Hence, an exponential regression function is selected to model SAT runtime for the Anti-SAT scheme.

For modules utilizing multiple locking schemes, the total SAT runtime is estimated as the sum of the individual runtime models for each scheme. Prior work [7] demonstrated that in compound locking, keys for SAT-weak schemes can usually be extracted separately from other locking schemes. This suggests that SAT runtime contributions from compound locking can be treated additively. For example, to model the SAT runtime of a module locked using three schemes SFLL [17], CASLock [18], and SLL [19] with key $k = (k_{SFLL}, k_{CASLock}, k_{SLL})$, we define the regression function in Eqn. 3. The functional form of each term (e.g., exponential, linear) is derived from theoretical complexity analyses in [20], [43], while the regression-derived constants $(A, \ldots, F)$ are fitted to match empirical data. In practice, $(A, \ldots, F)$ are per combinational module parameters learned from measurements on the target module instance. They are not shared across modules or designs. Because parameters are unique per module, circuit topology/size is captured implicitly by the fitted coefficients. This is important because the efficacy of locking and attack strategies varies significantly with circuit topology, making localized fitting necessary for accuracy.

$$SAT(k) = A*2^{B*|k_{SFLL}|-C} + D*2^{E*|k_{CASLock}|} + F*|k_{SLL}| \quad (3)$$

To adjust the constants in the SAT runtime model of Eqn. 3, we measure runtime data by subjecting the smallest key-length configurations (for each locking scheme) to a SAT attack, such as the open-source attack from [21]. Since these smaller key sizes yield lower SAT complexity, they are computationally more feasible to simulate, enabling the effective measurement of runtime to fit the model's parameters. Meanwhile, theoretical derivations indicate that SAT runtime grows predictably as key size increases, ensuring correct "tail" behavior for larger keys without requiring exhaustive simulation. This SAT runtime modeling strategy is evaluated through exhaustive simulation in Sec. VI-C, where we demonstrate that it achieves high accuracy (e.g., $(R^2 > 0.99)$) for the evaluated circuit and locking techniques, even at higher key lengths.

C. Iterative Selection of Simulation Points

Near-optimal regions in the design space based on design objectives are usually more important to a designer than those farther away, which are less likely to produce a high-quality solution. Hence, to efficiently navigate the large design space posed by the logic locking configuration problem, DSMLock performs directed simulation. Specifically, directed simulation is performed by focusing additional simulation and updates to the model towards predicted promising regions of the design space. We refer to these predicted favorable regions of the model as regions of interest (ROI). This ROI-based strategy is similar to methods employed in DSM for 3D ICs [11]. By generating additional simulation points within these promising regions of the design space, the model becomes more accurate in these regions.

The data points for model updates are selected as follows. From the complete design space Ω, the designer aims to discover a favorable configuration that meets specified objectives or cost functions. In this work, we consider four main objectives: the S metric (S), the SAT runtime (T), area overhead (A), and power overhead (P). Although we have focused on these four performance metrics, the approach is generalized and allows the designer to remove certain metrics or add new ones, depending on the system requirements. To formalize the directed simulation, we define the ROI using Eqn. 4, shown below. Here, (S_d, T_d, A_d, P_d) denote the values of these four objectives for the current best design d. Then, the ROI consists of all design points $i \in \Omega$ whose objective values (S_i, T_i, A_i, P_i) lie within a user-specified threshold $\Phi = \{\phi_S, \phi_T, \phi_A, \phi_P\}$. In each iteration, the algorithm selects η points from the ROI that exhibit the best-predicted performance. These points are simulated, and the resulting data is used to update the model in the next iteration. We then re-compute the ROI using the updated current best design (S_d, T_d, A_d, P_d) and repeat this loop until the search stops improving. This ROI-focused sampling approach ensures focused exploration of the most promising regions in the design space that meet the specified requirements. This ROI-based approach is drawn from prior DSM work, such as [11], which explores DSM for 3D-IC design.

$$ROI = \Bigg\{ i \in \Omega \;\Bigg|\; |S_d - S_i| \le \phi_S \wedge \left|\frac{P_d - P_i}{P_d}\right| \le \phi_P \wedge \left|\frac{A_d - A_i}{A_d}\right| \le \phi_A \wedge \left|\frac{T_i - T_d}{T_d}\right| \le \phi_T \Bigg\} \quad (4)$$

D. Optimization Techniques

To identify high-quality solutions while accounting for multiple design objectives and constraints, we adopt a heuristic approach using GenSA [45]. Unlike greedy approaches that optimize each critical module separately [13], we apply GenSA for the entire design space consisting of multiple modules. Hence, DSMLock considers an exponentially larger design space consisting of all valid locking solutions in a circuit, compared to prior module-level locking methodologies [17]–[19] and DSM-based locking approaches [13] which consider a small subset of possible locking configurations. GenSA balances local exploitation (improving upon the current solution) with global exploration (escaping local minima) [45]. This ensures module interactions are accounted for, enabling more effective system-level optimization.

We reinforce GenSA's search by incorporating penalty terms into the designer-defined cost function, which helps enforce constraints (e.g., area, security) and discourages infeasible solutions while still exploring the design space for high-quality configurations. For example, our objective function focuses on minimizing power overhead and includes penalty terms to discourage violations of critical constraints (e.g., maximum allowed security (S), area (A), or minimum SAT runtime (T)). In a scenario, if a candidate design exceeds any of the designer-specified limits ($S > S_{max}$, $A > A_{max}$, $T < T_{min}$, or $K > K_{target}$), GenSA imposes a large penalty on its cost, making it less likely to be selected as the new best solution. Although the example presented in this work focuses on minimizing power, DSMLock is not restricted to a specific optimization goal. In principle, the same penalty-based formulation can be adapted to any designer-defined key objective (e.g., minimizing area, maximizing security) by simply redefining the cost metric and adjusting constraint terms depending on application and designer needs. By combining the user-defined cost function with penalty terms at the optimization stage, DSMLock systematically balances the exploration of promising configurations against the need to discard those that break critical design limits. We define the overall cost function with these weighted penalties as follows:

$$\begin{aligned} f(x) = \alpha P(x) + \beta(S_{max} - S(x))^2[\text{if } S(x) > S_{max}] + \\ \gamma(A_{max} - A(x))^2[\text{if } A(x) > A_{max}] + \\ \delta(T_{min}^3 + T(x))[\text{if } T(x) < T_{min}] + \\ \omega[\text{if } |K| > K_{target}] \end{aligned} \quad (5)$$

Here, the coefficients α, β, γ, δ, and ω control the penalty magnitudes for power, security, area, SAT runtime, and key size, respectively.

E. Stopping Criteria

Stopping criteria determine when modeling halts. DSMLock considers two possible stopping criteria. 1) A sufficient portion of the design space was sampled to avoid model overfitting. This is user-defined, however, for the evaluation (Sec. VI), it is defined as 2% and 10% respectively for RISC-V and APRSC design space. 2) The ROI converges (i.e., the same η points are selected as the ROI in consecutive iterations). Here, the number for the consecutive iterations can be set by the designer depending on application requirements. Similar ROI for consecutive iterations indicates that the model has converged to a local minima. DSMLock also considers a minimum exploration period selected by the designer to prevent premature termination. Upon meeting the stopping criteria, DSMLock halts model updates, identifies the location of the favorable logic locking configuration for the design using the generated design space model, $f(m)$, and outputs this configuration along with the design objective metrics.

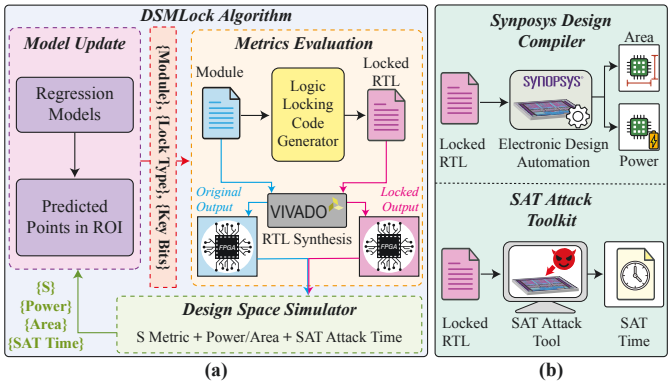

Fig. 6. (a) Overview of the DSMLock algorithm and simulation framework developed to evaluate DSM for system-level locking configuration. (b) Estimation of area and power using Synopsys Design Compiler, and SAT attack runtime using the open-source SAT attack from [21].

VI. Experimental Evaluation

We evaluate the proposed DSMLock algorithm through two experiments using implementations of a RISC-V processor and an APRSC communication interface [15], [16]. First, we explore RISC-V ALU implementation. In this setup, the design space for logic locking is sufficiently small to allow for exhaustive simulation. This enables us to compare different initial sampling methods (random vs. uniform) and regression models and to benchmark DSMLock's predictions against the exhaustive baseline. Next, we extend our evaluation by applying DSMLock to lock both the RISC-V processor and the APRSC interface using a designer-specified cost function to demonstrate the algorithm's flexibility. Finally, we compare the locking solutions produced by our model-based approach with those obtained through a conventional module-level locking method, highlighting differences in efficiency and security.

A. Experimental Setup and Simulation Flow

To evaluate DSMLock, we implemented the algorithm described in Section V. We have outlined the overall DSMLock infrastructure/flow developed for evaluation in Fig. 6. Each component of this setup is described below.

1) Design Objective (Cost Function): The designer-specified objective function (i.e., cost function) considered for this study is defined by Eqn. 6 and 7. The first cost function describes a scenario in which the designer aims to identify a candidate locking configuration (x) from the considered design space Ω that minimizes power overhead, $P(x)$, while satisfying other design objectives. In particular, the area overhead, defined as $A_o = A_{lock}/A_{no\_lock}$ must not exceed a predefined percentage of the original design (i.e., $A_{o(RISC-V)} = 3\%$ and $A_{o(APRSC)} = 10\%$), S-Metric must remain at or below 10^{-4} to ensure security, and the SAT attack runtime must exceed 7 days. The second cost function (Eqn. 7) represents a similar scenario, where the designer prioritizes minimizing area overhead $A(x)$, while ensuring the power overhead remains within 2% for both designs. The security and SAT runtime constraints remain unchanged. These constraints together define the feasible solution space for DSMLock. However, we again note that DSMLock can optimize for arbitrary designer-specified cost functions, as described in Section V-D.

$$\{x \in \Omega | [A_{o(RISC-V)}(x) \leq 3\% \vee A_{o(APRSC)}(x) \leq 10\%] \wedge S(x) \leq 10^{-4} \wedge T(x) > 7 \text{ days} \wedge min(P(x))\} \quad (6)$$

$$\{x \in \Omega | [P_{o(RISC-V,\ APRSC)}(x) \leq 3\%] \wedge S(x) \leq 10^{-4} \wedge T(x) > 7 \text{ days} \wedge min(A(x))\} \quad (7)$$

2) Weighted Penalty: DSMLock incorporates a penalty-based mechanism within GenSA to enforce hard constraints on four design variables: the security metric (S), area (A), SAT runtime (T), and key size (K). When a candidate design exceeds the designer-specified goal (see Sec. VI-A1), a large penalty term is added to the objective function to discourage infeasible solutions while still allowing exploration near constraint boundaries. The penalty weights were empirically tuned to reflect their relative importance in meeting the system design goals. In our case, minimizing power and area is the primary optimization objective; however, without strict constraints on key size and the S-Metric, the optimizer may choose configurations that appear secure but introduce excessive hardware overhead. Therefore, key size (ω) and security metric (β) are assigned higher weights to enforce security within acceptable cost, while power (α) and area (γ) are allowed greater flexibility. SAT runtime (δ) is weighted lowest, as the chosen locking techniques already ensure baseline SAT-resilience. The weighted penalty-based approach effectively discourages infeasible solutions while still allowing exploration near constraint boundaries. The specific penalty values used in this study are detailed in Eq. 8.

$$\begin{aligned} \alpha &= 10^4 \text{ (Power metrics weight)} \\ \beta &= 10^7 \text{ (S-Metric metrics weight)} \\ \gamma &= 10^4 \text{ (Area metrics weight)} \\ \delta &= 1 \text{ (SAT time metrics weight)} \\ \omega &= 10^{19} \text{ (Key size metrics weight)} \end{aligned} \quad (8)$$

3) DSMLock User-Specified Parameters: The initial models for the DSMLock are created from $\eta = 20$ selected points from the considered design space Ω. During each iteration, the $M = 5$ points with the lowest predicted cost are selected from the ROI to update the model. The ROI is defined using the radii for security ($\phi_s = 3 \times 10^{-5}$), timing ($\phi_t = 0.02$), area ($\phi_a = 0.05$), and power ($\phi_p = 0.05$). They define the acceptable deviation from the predicted optimal values when selecting candidate points for the next iteration, ensuring exploration of the most promising design space regions.

4) Logic Locking Configuration: For evaluation, we selected three logic locking schemes: SFLL [17], CASLock [18], and SLL [19]. These schemes were selected to provide a reasonable cross-section of locking strategies and present different security/overhead tradeoffs. However, the framework is technique-agnostic. Designers can incorporate any locking

techniques, including LUT-based [46], [47], FPGA-embedded approaches [48], [49], and RTL/HLS-based locking techniques [23], [24], [40], provided their methodology and performance metrics are available. We then construct a design space that allows each module to be locked with the designer-selected techniques and possible target key sizes. We note that this design space considers multiple candidate locking techniques implemented together, both within a single module and at the system level. This formulation, detailed in Section III, enables GenSA to explore the locking configuration design space to identify high-quality locking configurations capable of meeting design objectives. We implemented each locking scheme with an automated code generator that follows each technique as closely as possible to the approach outlined in the respective work to provide a realistic implementation in line with prior work [17]–[19].

5) System Security Metric (S): To estimate the S-Metric, we applied 50,000 test inputs to the processor, as outlined in Section IV-A. These test inputs were drawn from the unit tests covering realistic operating scenarios for each RISC-V and APRSC implementation [15], [16]. By subjecting both the locked and unlocked designs to this same input sequence, we systematically measured how frequently errors occurred (error frequency) and how long it took to recover (error severity), thereby quantifying S-Metric (see Eq. 1).

6) SAT Runtime: SAT attack runtime was calculated using the open-source SAT attack in [21].

7) Power/Area Overhead: Power, area, and timing for each circuit were measured with the Synopsys Design Compiler using the Synopsys 32nm SAED library. All evaluated designs were subject to a 100MHz clock constraint.

B. Robustness and Utility of the S-Metric

To demonstrate the robustness and utility of the proposed S-Metric, we consider a scenario where approximate keys with varying error rates are applied to a locked system. This is representative of approximate attack strategies such as appSAT [50] or Double DIP [51]. For this experiment, we locked an ALU, implemented in a RISC-V core, and identified approximate keys that achieve target output error rates of $e \in \{0.5\%, 1\%, 5\%, \text{ and } 10\%\}$. For each approximate key, we applied 500,000 random input stimuli to the RISC-V core and computed the module-level and system-level S-metrics, and compared them against the average error rate baseline.

Table II represents the output mismatch rate and S-Metric values at the module and system level with corresponding error rate. For the ALU, due to being a combinational circuit, any injected error is highly likely to appear immediately at the output. This is observed as the output mismatch rate closely follows the nominal error rate. However, for lower error rates ($\epsilon < 5\%$), at the system-level we see no erroneous outputs, which makes the S-metrics undefined (reported as infinity). This indicates that the ALU-level errors did not propagate through due to the inherent error resilience of the larger processor it resides in (e.g., pipeline and architectural masking). Hence, the S-metric demonstrates that an approximate key applied to the locking configuration with this corresponding error rate will not effectively prevent unauthorized use of the core, rendering it ineffective. As the error rate increases (5%–10%), the system-level S-metric becomes finite and decreases, revealing the point at which system functionality is meaningfully degraded. We note that at $\epsilon = 10\%$, we observe a higher error rate for the ALU design than normal. This happens because, in a stateful processor, it is possible that erroneous ALU results corrupt the internal state of the processor itself (e.g., a wrong program counter jump). As a result, future cycles execute from that bad state, cascading into further errors and driving the cumulative mismatch above the injected error rate. These results support the robustness of the S-metric compared to prior evaluation metrics, such as module error rate, which are unable to capture the complex interactions between locking-induced errors and the broader system that impact security, particularly in the case of approximate keys.

TABLE II
ROBUSTNESS OF THE S-METRIC UNDER VARYING ERROR RATE

Nominal error (%)	Output Mismatch (ALU) (%)	Output Mismatch (RISC-V) (%)	S-Metric (ALU)	S-Metric (RISC-V)
0.5%	0.501	0.00	197.86	inf.
1%	1.01	0.00	97.79	inf.
5%	6.07	2.62	15.49	37.07
10%	23.01	20.81	3.35	3.81

C. Exhaustive Characterization

To begin our evaluation, we aim to assess the fit of the models produced by the DSMLock algorithm. If the produced models are not predictive of unseen regions of the design space, they will not be effective in identifying favorable locking configurations. To do so, we conducted an exhaustive evaluation using a small test case where the entire design space could be characterized. Specifically, we focus on securing a RISC-V processor while considering ALU as the only critical IP to be protected, with a key length of 64 bits (i.e., K_{target} = 64). This setup results in approximately 905 potential locking combinations. Notably, even in this constrained test case, the simulation ran for over 100 hours. This simulation time supports the impracticality of exhaustive design space exploration and highlights the value of our model-driven optimization.

*1) **Experiment 1: Analysis of System-Wide Locking Design Space**:* Figure 7 represents the variation of the S-Metric, power, and area overhead across all possible locking

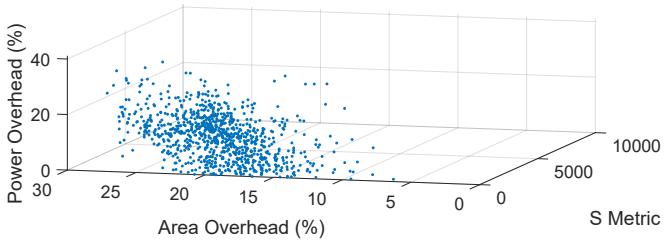

Fig. 7. Exhaustive characterization of the logic locking design space for a single locked module (RISC-V ALU).

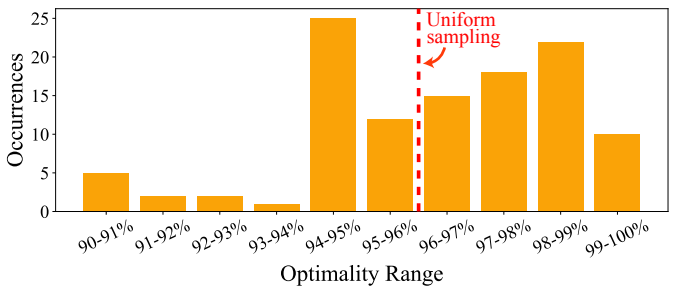

Fig. 8. Distribution of solution optimality achieved through random sampling, and highlighting the uniform sampling baseline (red-dashed line)

configurations for a RISC-V-based ALU implementation with a target key length of 64 bits. Based on these results, we make two observations: 1) The figure highlights the complex and non-trivial nature of the locking design space, even for a single locked module. This suggests the challenge of identifying favorable locking configurations in a design. 2) A majority of locking configurations are far from optimal, emphasizing the importance of locking configuration problems.

*2) **Experiment 2: Initial Design Space Sampling Methodology**:* For our DSMLock algorithm, we explored two possible initial design space sampling methodologies (uniform and random sampling) and measured the final solution quality by comparing the DSMLock-produced configuration to the globally optimal configuration (i.e., achieved through exhaustive simulation). When using uniform sampling, a single run yielded a final solution quality of approximately 96%. In contrast, random sampling (repeated the experiment 110 times to capture variance) yielded final solution quality ranging from about 90% to 99%, with a large cluster around 94–95%, and 55% of these runs exceeding 95.74% (see Fig. 8). While random sampling can occasionally find slightly better solutions, it requires multiple runs to achieve stable results, thereby increasing computational overhead. Since uniform sampling provides consistently high solution quality in a single run, we selected it for our subsequent evaluations.

*3) **Experiment 3: Model Fitting Performance Analysis**:* Building on the modeling strategies introduced in Section V-B, we conducted a comprehensive comparison of four model-fitting approaches for an exhaustively characterized RISC-V ALU with a 64-bit key budget. As summarized in Table III, SSANOVA demonstrates the highest accuracy for DSMLock's S-Metric, Power, and Area models. While conventional regression methods perform adequately for Power and Area, they struggle to capture the more complex S-Metric. By consistently achieving $R^2 > 0.99$ across all three metrics, SSANOVA proves to be the most robust approach and is thus selected for subsequent evaluations.

TABLE III
R^2 FOR DSMLOCK-PRODUCED MODELS IN THE RISC-V ALU.

Model	S-Metric	Power	Area
SSANOVA	0.9990	0.9990	0.9980
Linear	0.7145	0.9112	0.8590
Polynomial	0.8164	0.9226	0.9180
Exponential	0.9126	0.9206	0.8589

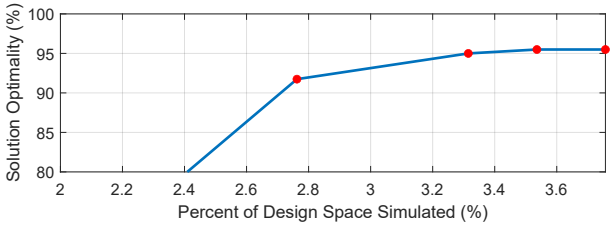

Fig. 9. Percent of design space simulated vs. the optimality of DSMLock-identified locking solution in RISC-V ALU.

*4) **Experiment 4: DSM-Predicted Solution Quality**:* To evaluate how DSMLock's solution quality changes with increasing design space coverage, we conducted an experiment on a RISC-V–based ALU design (target key length of 64 bits). We define optimality as p_{opt}/p_{pred}, where p_{opt} represents the power consumption of the best-known configuration discovered through exhaustive simulation, and p_{pred} denotes the power consumption of the configuration predicted by our DSMLock algorithm. In this context, values closer to 1 indicate promising performance. Fig. 9 plots the optimality of DSMLock's selected configurations against the percentage of the design space explored. As DSMLock samples more of the design space, solution quality generally improves because additional data reduces model uncertainty and guides the algorithm toward more effective configurations. Eventually, performance gains diminish, highlighting the importance of termination criteria to avoid excessive computation. In our experiment, DSMLock reached its termination conditions after sampling only 3.6% of the design space, identifying a configuration that achieved 96% optimality while meeting all imposed constraints as mentioned in Eqn. 6. The observation supports DSMLock's ability to discover promising locking solutions with less simulation overhead than exhaustive methods.

*5) **Experiment 5: Relationship Between Key Size and Design Overhead**:* To quantify the trade-offs between key size and the design overhead (Area, Power, and S-metric), we continued our simulation by exhaustively locking the ALU of a RISC-V processor with 765 different locking configurations, consisting of 3 locking techniques (SFLL, CASLock, SLL) and multiple key lengths (i.e., $K_{target} = \{8, 16, 24, 32, 40, 48, 56\}$). Figure 10 shows the complex relationship between key length and the measured overheads of the candidate locking configurations. Here, red markers connected by a blue line indicate the sample mean for each metric and key size, while the orange vertical bars show the min–max envelope. When grouping the data by key size, it becomes evident that increasing key length generally increases design overhead. However, the results show that there is substantial variance among locking configurations such that key length and design overhead cannot be easily related to one another. These results suggest that key size alone is not a reliable predictor of overhead or security, which motivates our DSMLock modeling-based, system-aware DSE.

D. Generalization to Alternative Attack Strategies

DSMLock is proposed as a threat-model-agnostic tool to configure locking in arbitrary designs. To be effective under

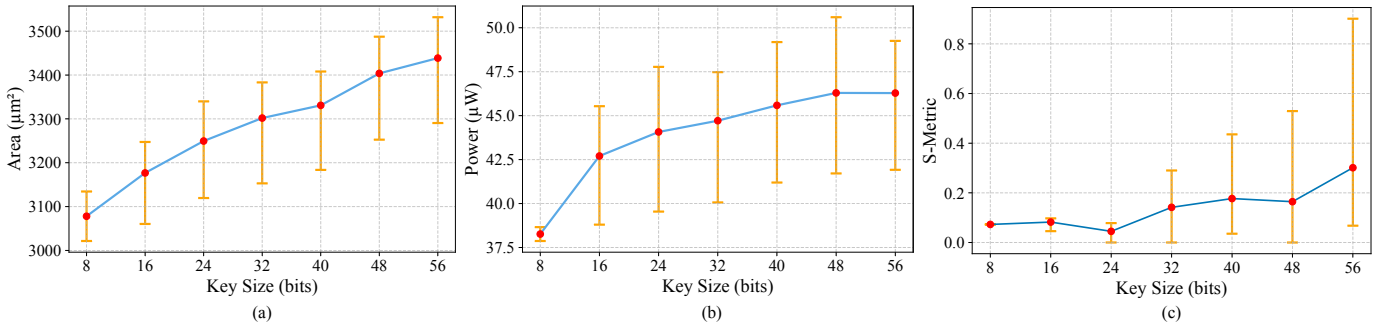

Fig. 10. Relationship between locking key size and (a) Area, (b) Power, and (c) S-Metric. The red circles denote the mean across locking configurations and orange vertical bars show the min–max range. The substantial inter-key overlaps indicate that key size alone is not a reliable predictor of overhead or security.

multiple threat models, DSMLock must accurately model varied attack strategies (i.e., it must generalize). To evaluate this, we considered the SWEEP attack [52], a structural oracle-less modeling attack, and evaluated whether DSMLock can produce a representative model that predicts its efficacy on RISC-V processor benchmark. To do so, we ran SWEEP on all 765 generated locking configurations of the exhaustively characterized RISC-V ALU with a 64-bit key budget, then used DSMLock to iteratively fit a model predicting SWEEP efficacy. Figure 11 shows the model's R^2 versus the percent of the design space sampled. Based on these results, DSMLock achieved $R^2 > 0.94$ after sampling only 3% of the design space. This indicates that DSMLock can accurately model attacks under alternative threat models, supporting the generalizability of the proposed modeling algorithms.

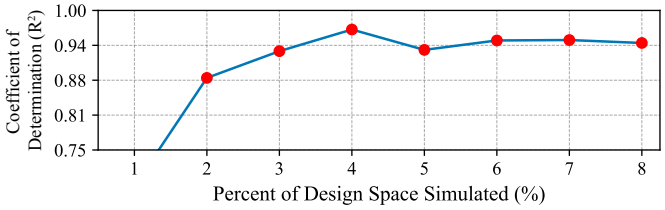

Fig. 11. Percent of design space simulated vs. the R^2 for DSMLock-generated model of SWEEP attack efficacy.

E. Full-System Evaluation

In this section, we consider a complete system-level locking configuration problem, where we secure the entire RISC-V core and APRSC interface [15], [16]. For both cases, the critical IP modules to be locked are listed in Table I. We have assigned a key length target of 160 bits for both systems and applied our proposed DSMLock algorithm to identify favorable locking configurations under two objective functions: a power-based cost function (Eqn. 6) and an area-based cost function (Eqn. 7). For comparison, we also implemented conventional module-level locking by selecting and applying three candidate logic locking schemes (SFLL [17], CASLock [18], and SLL [19]) to each module as described by the authors. To assess solution quality, we compare the locking solutions using four design metrics along with the cost functions outlined in Sec. VI-C.

1) Results and Analysis: Tables IV and V summarize the S-Metric, SAT runtime, total area, and power for each logic-locking approach under both power and area-based objective functions. Cells highlighted in green indicate that designer-specified constraints are satisfied, while red cells indicate failure to do so. We make the following observations based on these results:

Constraint Satisfaction: DSMLock meets all design constraints across both the RISC-V and APRSC designs for both cost functions. This is expected because traditional approaches typically optimize for module-level goals rather than system-level objectives.

TABLE IV
DESIGN OBJECTIVES FOR RISC-V LOCKING CONFIGURED BY DSMLOCK COMPARED TO CONVENTIONAL LOCKING APPROACHES. FOR THE POWER-FOCUSED COST FUNCTION, THE MAXIMUM ALLOWABLE AREA = BASE AREA $\times$ 1.03 = 6028.96, $\mu m^2 \times 1.03 = 6209.83, \mu m^2$. FOR THE AREA-FOCUSED COST FUNCTION, THE MAXIMUM ALLOWABLE POWER = BASE POWER $\times$ 1.03 = $900.9226\mu W \times 1.03 = 927.95\mu W$

Power Based Cost Function				
	DSMLock (RISC-V)	**SFLL [17]**	**CASLock [18]**	**SLL [19]**
S-Metric	2×10^{-5}	10000	10000	2×10^{-5}
SAT Runtime (s)	4.22×10^{28}	1.89×10^{31}	3.78×10^{9}	72
Area (μm^2)	6179.14	5967.71	6369.88	6325.62
Power (μW)	554.73	1270.37	890.38	829.39

Area Based Cost Function				
	DSMLock (RISC-V)	**SFLL [17]**	**CASLock [18]**	**SLL [19]**
S-Metric	4×10^{-6}	10000	10000	2×10^{-5}
SAT Runtime (s)	5.59×10^{9}	1.89×10^{31}	3.78×10^{9}	72
Area (μm^2)	5837.78	5967.71	6369.88	6325.62
Power (μW)	499.454	1270.37	890.38	829.39

TABLE V
DESIGN OBJECTIVES FOR APRSC LOCKING CONFIGURED BY DSMLOCK COMPARED TO CONVENTIONAL LOCKING APPROACHES. FOR THE POWER-FOCUSED COST FUNCTION, THE MAXIMUM ALLOWABLE AREA = BASE AREA $\times$ 1.10 = $25,628.74\mu m^2 \times 1.10 = 28,191.61, \mu m^2$. FOR THE AREA-FOCUSED COST FUNCTION, THE MAXIMUM ALLOWABLE POWER = BASE POWER $\times$ 1.03 = $34276.3678\mu W \times 1.03 = 35304.658\mu W$

Power Based Cost Function				
	DSMLock (APRSC)	**SFLL [17]**	**CASLock [18]**	**SLL [19]**
S-Metric	5.6×10^{-5}	3332.07	1000	2×10^{-5}
SAT Runtime (s)	6.35×10^{9}	2.8×10^{9}	2.31×10^{10}	44
Area (μm^2)	27476.81	29253.11	27984.58	28918.08
Power (μW)	35149.77	38511.76	36075.98	36914.72

Area Based Cost Function				
	DSMLock (APRSC)	**SFLL [17]**	**CASLock [18]**	**SLL [19]**
S-Metric	4×10^{-6}	3332.07	1000	2×10^{-5}
SAT Runtime (s)	4.78×10^{11}	2.8×10^{9}	2.31×10^{10}	44
Area (μm^2)	27753.237	29253.11	27984.58	28918.08
Power (μW)	35293.7473	38511.76	36075.98	36914.72

Power Efficiency: For the power-based cost function, the DSMLock-identified locking configuration requires 42.4% less power for RISC-V and 6.0% less power for APRSC than the solutions identified by conventional locking approaches on average. For the area-based cost function, the DSMLock-identified locking configurations are lower power than the conventional locking approaches for both RISC-V and APRSC-based implementations. Hence, in both designs, DSMLock consistently achieves lower power overhead than the conventional module-level locking benchmark strategies.

Area Efficiency: For the power-based cost function, the DSMLock-identified locking configurations are lower than the conventional locking approaches for both RISC-V and APRSC-based implementations. For the area-based cost function, the DSMLock-identified locking configuration requires 6.1% less area for the RISC-V and 3.3% less area for the APRSC than the solutions identified by conventional locking approaches on average. Hence, in both designs, DSMLock consistently achieves lower area overhead than the conventional module-level locking benchmark strategies.

These results indicate that our DSMLock algorithm outperforms the three evaluated conventional locking strategies by identifying solutions that meet system-level design objectives while substantially reducing target overhead constraints.

VII. CONCLUSION

We propose DSMLock to generate system-level models of the logic-locking design space for arbitrary IC by simulating only a small subset of the design space. Through selective sampling and simulation, DSMLock builds accurate predictive models that guide subsequent exploration, identifying promising locking solutions under designer-specified constraints on security, attack resilience, area overhead, and power. To evaluate DSMLock, we compare DSMLock-generated locking configurations against three module-level locking approaches for a RISC-V processor and an APRSC communication interface. Under a power-focused cost function, DSMLock achieved average power reductions of 42.4% and 6.0% for the locked modules in the RISC-V and APRSC designs, respectively. Under an area-focused cost function, DSMLock achieved average area reductions of 6.1% for the RISC-V locked modules and 3.3% for the APRSC locked modules. These results demonstrate DSMLock's ability to model the complex logic locking design space and identify high-quality locking configurations that optimize for designer-specified objectives.

REFERENCES

[1] K. Zamiri Azar, H. Mardani Kamali, F. Farahmandi, and M. Tehranipoor, "Ip protection: A historical view," in *Understanding Logic Locking*. Springer, 2023, pp. 47–62.

[2] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[3] A. Chakraborty, N. G. Jayasankaran, Y. Liu, J. Rajendran, O. Sinanoglu, A. Srivastava, Y. Xie, M. Yasin, and M. Zuzak, "Keynote: A disquisition on logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 1952–1972, 2019.

[4] K. Z. Azar, H. M. Kamali, F. Farahmandi, and M. Tehranipoor, *Understanding Logic Locking*. Springer, 2024.

[5] M. Zuzak and A. Srivastava, "Obfusgem: Enhancing processor design obfuscation through security-aware on-chip memory and data path design," in *Proceedings of the International Symposium on Memory Systems*, 2020, pp. 260–271.

[6] K. Shamsi, D. Z. Pan, and Y. Jin, "On the impossibility of approximation-resilient circuit locking," in *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2019, pp. 161–170.

[7] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "Appsat: Approximately deobfuscating integrated circuits," in *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2017, pp. 95–100.

[8] M. Yasin, C. Zhao, and J. J. Rajendran, "Sfll-hls: Stripped-functionality logic locking meets high-level synthesis," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–4.

[9] M. Zuzak, Y. Liu, and A. Srivastava, "A resource binding approach to logic obfuscation," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 235–240.

[10] W. Jia, K. A. Shaw, and M. Martonosi, "Stargazer: Automated regression-based gpu design space exploration," in *2012 IEEE International Symposium on Performance Analysis of Systems & Software*. IEEE, 2012, pp. 2–13.

[11] C. Serafy, Z. Yang, and A. Srivastava, "Design space modeling and simulation for physically constrained 3d cpus," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, 2017, pp. 375–380.

[12] M. Tehranipoor, K. Zamiri Azar, N. Asadizanjani, F. Rahman, H. Mardani Kamali, and F. Farahmandi, "Advances in logic locking," in *Hardware Security: A Look into the Future*. Springer, 2024, pp. 53–142.

[13] L. Lam, M. Melnyk, and M. Zuzak, "Low overhead logic locking for system-level security: A design space modeling approach," in *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design*, 2024, pp. 1–6.

[14] M. Zuzak, "Dsm for logic locking," https://github.com/mzuzak/DSM-for-Logic-Locking, 2020, gitHub repository, accessed: 2025-03-26.

[15] ultraembedded, "BiRISC-V: A Binary Translation RISC-V Core," https://github.com/ultraembedded/biriscv, 2020, accessed: Jan. 2025.

[16] J. McCanny, D. Ridge, Y. Hu, and J. Hunter, "Hierarchical vhdl libraries for dsp asic design," in *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1. IEEE, 1997, pp. 675–678.

[17] A. Sengupta, M. Nabeel, N. Limaye, M. Ashraf, and O. Sinanoglu, "Truly stripping functionality for logic locking: A fault-based perspective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4439–4452, 2020.

[18] B. Shakya, X. Xu, M. Tehranipoor, and D. Forte, "Cas-lock: A security-corruptibility trade-off resilient logic locking scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 175–202, 2020.

[19] M. Yasin, J. J. Rajendran, O. Sinanoglu, and R. Karri, "On improving the security of logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 9, pp. 1411–1424, 2015.

[20] M. Zuzak, Y. Liu, and A. Srivastava, "Trace logic locking: Improving the parametric space of logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 8, pp. 1531–1544, 2020.

[21] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2015, pp. 137–143.

[22] A. K. T. Moosa, B. Tan, and R. Karri, "Scaling attacks on large logic-locked designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[23] H. M. Kamali, K. Z. Azar, F. Farahmandi, and M. Tehranipoor, "Shell: Shrinking efpga fabrics for logic locking," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.

[24] M. R. Muttaki, S. Saha, H. M. Kamali, F. Rahman, M. Tehranipoor, and F. Farahmandi, "Rtlock: Ip protection using scan-aware logic locking at rtl," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.

[25] M. R. Muttaki, R. Mohammadivojdan, H. M. Kamali, M. Tehranipoor, and F. Farahmandi, "Hlock+: A robust and low-overhead logic locking at the high-level language," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 7, pp. 2149–2162, 2022.

[26] B. Shakya, N. Asadizanjani, D. Forte, and M. Tehranipoor, "Chip editor: leveraging circuit edit for logic obfuscation and trusted fabrication," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2016, pp. 1–8.

[27] J. Gandhi, N. Handa, A. Nayak, D. Shekhawat, M. Santosh, J. Dofe, and J. G. Pandey, "Shakti: Securing hardware ips by cascade gated multiplexer-based logic obfuscation," in *2025 38th International Conference on VLSI Design and 2024 23rd International Conference on Embedded Systems (VLSID)*. IEEE, 2025, pp. 139–144.

[28] G. Ascia, V. Catania, A. G. Di Nuovo, M. Palesi, and D. Patti, "Efficient design space exploration for application specific systems-on-a-chip," *Journal of Systems Architecture*, vol. 53, no. 10, pp. 733–750, 2007.

[29] C. Pilato, L. Collini, L. Cassano, D. Sciuto, S. Garg, and R. Karri, "Optimizing the use of behavioral locking for high-level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 2, pp. 462–472, 2022.

[30] C. Silvano, W. Fornaciari, G. Palermo, V. Zaccaria, F. Castro, M. Martinez, S. Bocchio, R. Zafalon, P. Avasare, G. Vanmeerbeeck *et al.*, "Multicube: Multi-objective design space exploration of multi-core architectures," in *VLSI 2010 Annual Symposium: Selected papers*. Springer, 2011, pp. 47–63.

[31] Y. Xie, G. H. Loh, B. Black, and K. Bernstein, "Design space exploration for 3d architectures," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 2, no. 2, pp. 65–103, 2006.

[32] Y. Xie and Y. Ma, "Design space exploration for 3d integrated circuits," in *2008 9th International Conference on Solid-State and Integrated-Circuit Technology*. IEEE, 2008, pp. 2317–2320.

[33] E. Kang, E. Jackson, and W. Schulte, "An approach for effective design space exploration," in *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems: 16th Monterey Workshop 2010, Redmond, WA, USA, March 31-April 2, 2010, Revised Selected Papers 16*. Springer, 2011, pp. 33–54.

[34] A. D. Pimentel, "Exploring exploration: A tutorial introduction to embedded systems design space exploration," *IEEE Design & Test*, vol. 34, no. 1, pp. 77–90, 2016.

[35] K. Lahiri, A. Raghunathan, and S. Dey, "Design space exploration for optimizing on-chip communication architectures," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 23, no. 6, pp. 952–961, 2004.

[36] T. Taghavi, M. Thompson, and A. D. Pimentel, "Visualization of computer architecture simulation data for system-level design space exploration," in *Embedded Computer Systems: Architectures, Modeling, and Simulation: 9th International Workshop, SAMOS 2009, Samos, Greece, July 20-23, 2009. Proceedings 9*. Springer, 2009, pp. 149–160.

[37] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "From cryptography to logic locking: A survey on the architecture evolution of secure scan chains," *IEEE Access*, vol. 9, pp. 73 133–73 151, 2021.

[38] L. Collini, R. Karri, and C. Pilato, "A composable design space exploration framework to optimize behavioral locking," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 1359–1364.

[39] C. Pilato, F. Regazzoni, R. Karri, and S. Garg, "Tao: Techniques for algorithm-level obfuscation during high-level synthesis," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[40] M. R. Muttaki, R. Mohammadivojdan, H. M. Kamali, M. Tehranipoor, and F. Farahmandi, "Hlock+: A robust and low-overhead logic locking at the high-level language," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 7, pp. 2149–2162, 2022.

[41] M. Zuzak, Y. Liu, and A. Srivastava, "A resource binding approach to logic obfuscation," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 235–240.

[42] C. Gu, *Smoothing Spline ANOVA Models*. Springer Science & Business Media, 2013, vol. 297.

[43] H. Zhou et al, "Resolving the trilemma in logic encryption," in *IEEE/ACM International Conference on CAD (ICCAD)*, 2019.

[44] Y. Xie and A. Srivastava, "Anti-sat: Mitigating sat attack on logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 2, pp. 199–207, 2018.

[45] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[46] G. Kolhe, H. M. Kamali, M. Naicker, T. D. Sheaves, H. Mahmoodi, P. S. Manoj, H. Homayoun, S. Rafatirad, and A. Sasan, "Security and complexity analysis of lut-based obfuscation: From blueprint to reality," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.

[47] H. M. Kamali, K. Z. Azar, K. Gaj, H. Homayoun, and A. Sasan, "Lut-lock: A novel lut-based logic obfuscation for fpga-bitstream and asic-hardware protection," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2018, pp. 405–410.

[48] J. Bhandari, A. K. T. Moosa, B. Tan, C. Pilato, G. Gore, X. Tang, S. Temple, P.-E. Gaillardon, and R. Karri, "Exploring efpga-based redaction for ip protection," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[49] B. Hu, J. Tian, M. Shihab, G. R. Reddy, W. Swartz, Y. Makris, B. C. Schaefer, and C. Sechen, "Functional obfuscation of hardware accelerators through selective partial design extraction onto an embedded fpga," in *Proceedings of the 2019 Great Lakes Symposium on VLSI*, 2019, pp. 171–176.

[50] K. Shamsi, T. Meade, M. Li, D. Z. Pan, and Y. Jin, "On the approximation resiliency of logic locking and ic camouflaging schemes," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 2, pp. 347–359, 2018.

[51] Y. Shen and H. Zhou, "Double dip: Re-evaluating security of logic encryption algorithms," in *Proceedings of the Great Lakes Symposium on VLSI 2017*, 2017, pp. 179–184.

[52] A. Alaql, D. Forte, and S. Bhunia, "Sweep to the secret: A constant propagation attack on logic locking," in *2019 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2019, pp. 1–6.

Robi Paul (Student Member, IEEE) received his B.S. degree from the School of Electrical Engineering, Shahjalal University of Science and Technology, Bangladesh, in 2023. He is currently pursuing his Ph.D. degree with the Department of Computer Engineering, Rochester Institute of Technology, Rochester, USA. His current research interests include hardware security, focusing on integrity violation detection in machine learning models.

Long Lam (Student Member, IEEE) received his B.S. and M.S. degrees from the School of Computer Engineering, Rochester Institute of Technology, Rochester, USA, in 2024. He is currently a SoC Design Engineer at OmniVision Technology, Santa Clara, CA, USA. His current research interests include reconfigurable computing and image signal processing.

Maksym Melnyk (Student Member, IEEE) received the B.S. degree in Computer Engineering from the Rochester Institute of Technology, Rochester, NY, USA, in 2024. He is currently pursuing his Ph.D. degree with the Department of Computer Engineering, Rochester Institute of Technology, Rochester, USA. His research interests include hardware security, with a focus on logic locking, detection and mitigation of hardware Trojans, and secure design methodologies for integrated circuits.

Michael Zuzak (Member, IEEE) received the Ph.D. degree in electrical engineering from the University of Maryland, College Park, MD, USA, in 2022. He is an Assistant Professor with the Department of Computer Engineering, Rochester Institute of Technology, Rochester, NY, USA. His current research interests include hardware security, computer architecture, and electronic design automation.