RTLock: IP Protection using Scan-Aware Logic Locking at RTL

Md Rafid Muttaki, Shuvagata Saha, Hadi M Kamali, Fahim Rahman, Mark Tehranipoor and Farimah Farahmandi
Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, USA.
{m.muttaki, sh.saha, h.mardanikamali}@ufl.edu, {fahimrahman, tehranipoor, farimah}@ece.ufl.edu

***Abstract*—Conventional logic locking techniques mainly focus on gate-level netlists to combat IP piracy and IC overproduction. However, this is generally not sufficient for protecting semantics and behaviors of the design. Further, these techniques are even more objectionable when the IC supply chain is at risk of insider threats. This paper proposes RTLock, a robust logic locking framework at the RTL abstraction. RTLock provides a detailed formal analysis of the design specs at the RTL that determines the locking candidate points w.r.t. attacks resiliency (SAT/BMC), locking key size, and overhead. RTLock incorporates (partial) DFT infrastructure (scan chain) at the RTL, enabled with a scan locking mechanism. It allows us to push all the necessary security-driven actions to the highest abstraction level, thus making the flow EDA tool agnostic. Additionally, RTLock demonstrates why RTL-based locking must be coupled with encryption and management protocols (e.g., IEEE P1735), to be effective against insider threats. Our experimental results show that, vs. other techniques, RTLock protects the design against broader threats at low overhead and without compromising testability.**

***Index Terms*—Logic Locking, RTL, Design-for-Testability, Encryption.**

I. Introduction

The ever-increasing cost/complexity of IC manufacturing/testing and technology shrinking have pushed semiconductor industries to follow the horizontal business model, in which different IC supply chain steps are outsourced. However, potential untrustworthy entities may impose security threats, e.g., IP piracy and IC overproduction [1]. To protect designs against such vulnerabilities, logic locking is known as a proactive countermeasures, which has received significant attention in recent years [2]. Logic locking enables the post-fabrication activation using some extra gates, known as key gates, where the correct key recovers the original functionality. The vast majority of existing logic locking solutions have been proposed and implemented as a post-synthesis process on the gate-level netlist, e.g., EPIC [3]. However, with the emergence of satisfiability-based (SAT) attacks [4], all these techniques are already broken in a very short attack time. Although numerous countermeasures were introduced to thwart this attack [5]–[11]. However, these techniques also suffer shortcomings, such as low output corruptibility, high overhead, burdensome implementation challenges, and testability issues. Also, such shortcomings led to emerging new attacks that undermine the resiliency of these approaches [12]–[20].

More recently, few studies have investigated logic locking implemented at higher levels of abstractions, such as high-level-synthesis (HLS) [21]–[24], and particularly register-transfer level (RTL) [25]–[27]. The preliminary investigation has shown that synthesizing the locked RTL (or HLL) may provide superior resiliency against ML-based attacks. However, more recent studies reveal successful invocation of SMT and ML-based attacks on these locking techniques at RTL [27], [28]. Additionally, the definition of threat models might face significant changes over time invalidating all previous security solutions. For instance, insider threats can be realized in modern companies' environments in different forms, such as (i) a rogue employee at a design house as a malicious actor or (ii) illegal access to a company proprietary through a compromised cloud/network infrastructure. In such scenarios, plain information sharing could cause companies' sensitive information breaches. Hence, from the design team to verification, nearly any individual working within any trusted entity can be regarded as a threat, which invalidates gate-level locking.

The main goal of the existing HLS/RTL-based logic locking techniques is to protect the semantics/behavior of the design. But these methods do not include insider threats in their threat model. However, insider threats as an alarming issue, can leak secrets to an untrusted entity. This paper proposes *RTLock* as a comprehensive RTL-based logic locking framework to address this shortcoming in the previous threat models. In this framework, we push all actions related to the locking flow towards RTL. *RTLock* applies the locking to all parts of the RTL, from the operations to state machines. *RTLock* analyzes the RTL to determine the best locking positions based on the design specifications. Moreover, *RTLock* selects the candidates so that once the locked RTL goes through the synthesis, a more efficient distribution of locking key gates is acquired. Also, with insider threats, as (i) the locked RTL goes through the verification process and synthesis, and (ii) the gate-level design goes through post-synthesis transformations, we show how coupling IEEE P1735 standard into *RTLock* minimizes the possibility of information breach throughout IC supply chain steps. By doing so, in comparison with other RTL locking techniques, as shown in Table I, *RTLock* would support the design against broader threats.

The main contributions of this work are summarized as follows: (1) *RTLock* is a locking framework at RTL that addresses locking as an optimization problem relying on integer linear programming (ILP). *RTLock* is able to provide the best possible solution within given constraints in terms of the desired attack resiliency (SAT/BMC), key size, and overhead. (2) *RTLock* is enabled with a novel partial scan insertion+locking mechanism at RTL representation, allowing designs to be fully locked in

TABLE I: High-level Comparison of RTL-based Logic Locking Techniques

Technique	Against IP Piracy by ↓			Broken by
	Insider Threats	Oracle Less	Oracle guided	
ASSURE [25]	✗	✓	✗	SAT [4], ML-based [27]
ASSURE + Scan [26]	✗	✓	✓	ML-based [27]
ML-resilient ASSURE [27]	✗	✓	✗	SAT [4]
proposed *RTLock*	✓	✓	✓	–

978-3-9819263-7-8/DATE23/© 2023 EDAA

further steps. It has been done via topological relating scan FFs and the key gates (+ the semantic/behavior of the design). (3) We integrate *RTLock* with IEEE P1735 which allows protecting the design against potential design house (insider) threats, e.g., through the verification stage. (4) We provide detailed security, testability, and overhead analysis of the *RTLock* framework.

II. Background

A. Threat Model

As shown in Fig. 1, different design-for-trust solutions have tried to thwart IC supply chain threats. However, each practice covers a part of threats throughout the IC supply chain. **IEEE P1735** provides *encryption and management* of IPs mainly at RTL [29] (see Fig. 1(a)). It is typically developed to address the threats originating from integration/verification to synthesis and does not consider threats from the further stages. **Gate-level logic locking**, as a post-synthesis process shown in 1(b), adds protection against untrusted entities in the fabrication and testing stages. However, with potential insider threats, the design is plain and ready to be reused for all preceding steps. **RTL-based logic locking** migrates the key-based augmentation to a higher abstraction layer. However, none of the existing RTL-based locking techniques offer any solution against insider threats (see Fig. 1(c)). Also, depending on the type of locking, the test engineer might needed to be trusted, which makes the locking technique more limited [2].

In *RTLock*, we assume that every entity in the design process is untrusted. Hence, the logic locking is done at the earliest design stage (after RTL design). We also assume that both broad categories of logic locking threat models, i.e., oracle-less and oracle-guided, can be applied to *RTLock* [4], [12]. In the traditional oracle-less threat model, the adversary has only access to a reverse-engineered netlist with all relative information, like the location of key gates, locking algorithms, etc. The adversary can also access the locked RTL, considering an insider threat exists. Conversely, in the oracle-guided model, the adversary has access to a reverse-engineered netlist (or locked RTL), and an unlocked/activated working IC. In this scenario, as shown in Fig. 1(d), every individual throughout the IC supply chain is untrusted. We elaborate further in Section III on how *RTLock* addresses these threats altogether.

B. Logic Locking: Post- vs. Pre-Synthesis

Logic locking can be applied at different levels of abstraction, from design specification to transistor level. The ultimate goal is to add a level of secrecy to the design, via key, to make its operation dependent on this secret. This secret is known and can only be initiated in a trusted entity and generally will be loaded into a tamper-proof memory (TPM). Conventionally, logic locking has been accomplished as a post-synthesis step of the design, via key-based XOR, MUX, and LUT key gates. For multiple reasons, these gate-level logic locking techniques are no longer desirable: (i) it is applied on an already-optimized netlist, where the semantic/behavior might be flattened/absorbed and could be vulnerable to structural-based analyses [30]. (ii) It does not provide any advantage (security) for the design team against probable threats within supposedly trusted entities. For instance, as an insider threat, a rogue employee might have access to the RTL design, which is not locked. From the design team perspective, it is more desirable to have the logic locking more correlated to the highly sensitive and

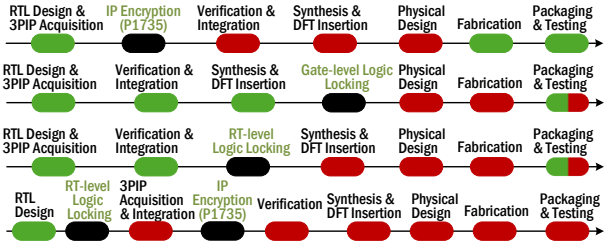

Fig. 1: Threat Models for: (a) IP Encryption and Management by IEEE P1735 [29], (b) Gate-level Logic Locking [3], [7], [9]–[11], [31], (c) RTL-based Logic Locking [25]–[27], (d) Fully Untrusted (Zero-Trust) in *RTLock*.

critical operations. However, this becomes impossible due to the flattening, merging, and other optimization heuristics applied during synthesis. RTL-based logic locking can facilitate the designer to resolve these issues altogether. RTL-based logic locking commonly consists of control flow (FSM), arithmetic, and constant locking used in the existing RTL-based techniques [25]–[27]. However, as reflected in Table I, each has just tried to concentrate on a specific threat model, which significantly restricts their applicability and efficiency.

III. Proposed Scheme: RTLock

In *RTLock*, three main steps are integrated into the regular IC supply chain flow. Fig. 2 shows the main steps of the IC supply chain equipped with locking through *RTLock*, where these three steps are highlighted in green. Step 1 is our novel scan-aware **functional RTL locking**, allowing us to apply all locking-oriented modifications on the RTL. Step 2 is the **P1735-based RTL encryption**, which will be done after the design integration. We demonstrate that P1735 is a must as part of the solution against insider threats, as it supports the verification fully *encrypted*. Since *RTLock* is a scan-aware RTL locking, to ensure maximum test/fault coverage, manual scan inserted+locked at RTL is integrated with other DFT structures added by the DFT synthesis in step 3. The following discusses the details of these three steps, as shown in Fig. 4:

A. Locking Procedure in RTLock

The locking part of *RTLock* includes *seven* major steps to transform a hardware description language (HDL) design into a functionally locked and scan-chain enabled+locked RTL. In the framework, we consider both the design and performance parameters to lock a design effectively. A set of design assets, critical locations, and operations are considered for design parameters. These performance parameters are related to the effectiveness of locking against SAT/BMC attacks and the overheads incurred during this process. Additionally, the framework facilitates the designer to choose the locking case combinations that satisfy the specification in terms of power, performance, and area (PPA) analysis. The following describes the steps:

(1) Analyzing the RTL: Initially the designer sets a specification list based on the design and performance parameters to be achieved. This list works as the input to an optimization tool (described in step 4) to efficiently choose locking candidates. In this step, the designer tracks the assets, critical operations, functionalities, and structures like state transitions of the design. For extracting the design's finite state machine (FSM), an in-house open-source tool [32] is used. The tool automatically identifies the control FSM of the design in text/graph format. For tracking the flow of assets, operations, and states, the design's control data flow graph (CDFG) is analyzed using the JasperGold formal verification tool.

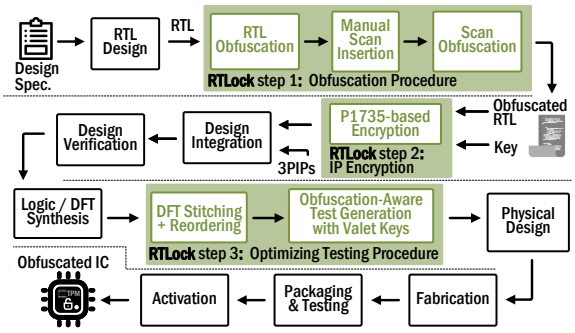

Fig. 2: RTLock-enabled IC Supply Chain.

(2) Selecting Locking Candidates: Based on the selected locking points in the design, *RTLock* supports three major classes of locking candidates at the RTL abstraction:

• *Constant Locking*: Constants of the design can be locked in various ways. One way is to use Boolean functions to relate the constant value and the locking key. For example, by $XOR/XNOR$ing each bit of the constant with a single key-bit. Alternatively, *RTLock* can substitute the original value with a key value stored in the design.

• *Arithmetic Operations Locking*: Arithmetic operations can also be locked in multiple ways. Locking key bits can be evaluated in conditional statements executing incorrect functionality for each wrong key condition. Note that, for the pairing of operators, *RTLock* has done it in a uniform and specific distribution (e.g., $+$ is only paired with $-$), making it resilient against ML-based operator-wise attack [27]. Another way to lock is to use certain key bits as the operand itself.

• *FSM-based Locking*: *RTLock* extends the locking candidates at RTL by extracting and targeting the state machine of the design. The designer can add locking-oriented augmentation on the control flow that adds protection for the asset transition/propagation, the flow of critical operations, etc. Some of the utilized methods in *RTLock* for the generic FSM locking scenarios (Fig. 3) are as follows:

(i) *Initialization Locking*: Applying the incorrect key can force the design's state machine to return to or stay in the initialization state rather than move to the supposedly next state. As a result, the output register shows the initialized values. As an example (3(b)), while applying the incorrect key, the FSM will keep looping in the state "$init\ lock$" through transition "t_{2F}".

(ii) *Incorrect State Transition*: When applying incorrect keys, the design can deviate from the expected sequence of state transitions. For example, the actual sequence of state machine shown in Fig. 3(a) would be "$t_1 \rightarrow t_2 \rightarrow t3 \rightarrow t4$". However, using this method, the traversal can be manipulated as shown in Fig. 3(c), in which the state "$init$" is bypassed.

(iii) *Skipping State*: Each state of the FSM contributes to the correct functionality of the design. Skipping any intermediate states will result in malfunction and output corruptibility. This scenario can be created by applying the locking key condition at the next state assignment. So, with one or more states eluded, higher output corruptibility will happen. Fig. 3(d) shows an example of a skipping state in which "$init$" has been skipped.

(iv) *Bypassing State*: One or more states can be created with dummy statements to facilitate locking. These fake states can replace the actual state transition. For example, as shown in Fig. 3(e), let us consider state "$bypass$" as the fake state, and when the key is wrong, the state traversal becomes "$idle \rightarrow bypass \rightarrow next$" instead of "$idle \rightarrow init \rightarrow idle \rightarrow next$".

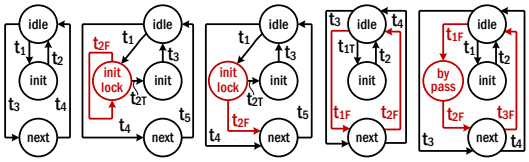

Fig. 3: Typical and Generic Case Studies for FSM Locking in *RTLock*.

(v) *Locking Inherent Signals*: Each state in the FSM is responsible for a set of signal assignments. Changing the signal assignment can lead to incorrect initialization of variables and wrong state traversal. The signal assignments within the FSM states can be locked to protect assets and states. Each instance of FSM locking can be executed with a single key bit.

(3) Database Creation: In this step, performance metrics for locking cases are evaluated and stored. Each locking case consists of a locking point and a candidate. For each locking point, there may be more than one locking candidate. For example, a constant can be locked by adding fake arithmetic or Boolean operations. *RTLock* considers these scenarios as individual cases. The framework synthesizes the locked design cases into gate-level netlists to analyze the performance parameters based on the attack resilience (SAT/BMC time) and incurred area overhead. This process stores information on the case number index, key size, overhead, and attack resilience (CPU time) and creates a database for all the cases. The database creation for each locking case is done offline.

(4) Selection of Cases: After database creation, an optimization problem arises to efficiently choose the best combination of the available cases from the database. For this, we utilize integer linear programming (ILP) [33] to generate reliable solutions for the optimization problem. For our case, the designer performance specification works as constraints, and the database cases work as inputs to the ILP tool. The locking candidates for a design are selected by solving the following ILP constraints/objectives:

$$\sum_{i=1}^{n} T_i C_i + (\%added\ Res.) \geq T_{spec} \quad \sum_{i=1}^{n} A_i C_i - (\%shared\ Ov.) \leq A_{spec} \tag{1}$$

$$\sum_{j=1}^{x} C_{1j} \leq 1; \sum_{j=1}^{x} C_{2j} \leq 1; \ ..... \ \sum_{j=1}^{x} C_{mj} \leq 1 \qquad min\ F = \sum_{i} C_i \tag{2}$$

Equation 1 describes the attack resilience and area overhead constraints for the locking cases. Here, T_i is the attack resilience and A_i is the area overhead from case i from the candidate list of m cases. T_{spec} and A_{spec} define the designer specification for the attack resilience and area overhead, respectively. We introduce two parameters, i.e., (*added Res.*) to model the increase in the attack resilience while merging multiple cases and (*shared Ov.*) to model the shared hardware resources leading to reduced area overhead, respectively. We set the percentage to (10-20)% depending on various designs. In the left part of Equation 2, C_{ij} denotes locking cases for locking point j and ensures that mutual exclusiveness is achieved. The right side of Equation 2 describes the objective function as the minimum number of chosen candidates satisfying the stated constraint equations. The output of the ILP tool is the 0/1 binary selection of locking candidates and the final locking key size is the summation of chosen candidates' key sizes.

(5) Update RTL: With the selected locking candidates at hand, the next step is to combine the candidates to build the

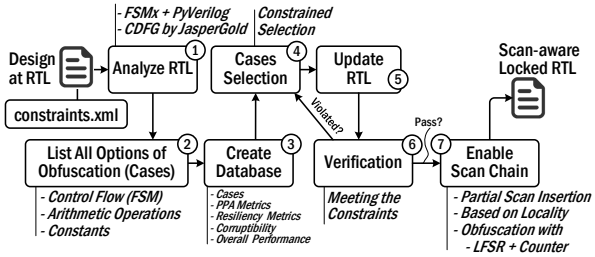

Fig. 4: Main Steps of *RTLock* for Scan-Aware RTL Locking.

functionally locked RTL. This step can be revisited multiple times in case the designer's specifications are not met yet.

(6) Design Verification: In this step, the locked design is evaluated based on the defined parameters (overheads, attack resilience, and output corruptibility). This step checks whether the input specification for the design is met and the locking acts as expected while applying correct/incorrect key combinations. The functional verification (IP-level) can be performed with a simulation-based approach or a more exhaustive logical equivalence checking by formal verification. If the verification fails (specification-wise), the design goes back to step 4.

(7) Partial Scan Chain Insertion + Locking: *RTLock* performs a partial locking oriented scan chain insertion at the RTL to enhance resiliency against oracle-guided attacks and push the security-oriented activities towards RTL, as full scan insertion manually can incur high overhead compared to industry level DFT synthesis tools. From the SCOAP analysis presented in [34], it is evident that the extraction of keys can be made more difficult by the low observability of the key-related registers. Based on this rule, we choose the candidates for the partial RTL scan registers within n combinational logic levels from the key inputs, where n is an arbitrary number. We select the scan locking key size based on the candidate registers and the overhead specification. Additionally, inspired by [11], we utilize a counter-LFSR-based scan locking. After this step, the augmented RTL is ready for the next design steps.

B. IP Encryption for Integration/Verification

To be resilient against insider threats, logic locking at the earliest design stage might be desirable. However, logic locking before verification becomes challenging as the verification requires a white-box definition of each module. Hence, to mitigate insider threats through RTL-based logic locking, IP encryption-based transformation is required that can be automated by a tool or processed by a trusted group in the IP design house [35]. The main aim here is to protect the locked IP against insider threats in the integration/verification team. Since the high-value IP under consideration is already locked, the encryption will be accomplished on a wrapped version of the IP fed by the logic locking key. Note that it is inevitable that the encryption methodology must rely on state-of-the-art recommendations such as using authenticated encryption with associated data (AEAD) [36], and the latest edition of standards like IEEE P1735 [29] must be engaged. After encryption, to access the locked IP, the integration/verification engineers, with provided private secret to the desired tool, such as Synopsys Synplify/Verdi or Cadence IES, gain the permission to operate on black-box IP design data without the possibility of either pirating the locked IP or retrieving the locking key.

C. Optimizing Testing Procedure

Since scan chain insertion+locking has been done partially at RTL, as a post-synthesis step, scan chains for the rest of the registers must be inserted at DFT synthesis. However, this decouples the scan chain corresponding to the pre-synthesis from the part inserted by the DFT tool. Hence, *stitching* of these two parts must be done. *RTLock* accomplishes this step effortlessly (just by connecting chains to build a longer chain). Additionally, as industry-level DFT compilers add the scan chain in an optimum way, hybrid manual+automatic scan chain insertion in *RTLock* may have a significant impact on PPA metrics. Hence, supported by the commercial DFT, scan chain *re-ordering* will be done for these parts. Using such a mechanism reduces the overhead of manual scan insertion.

IV. Security Assessment of RTLock

The main purpose of *RTLock* is to prevent IP piracy against a wider threat model, including insider threats. Although the combination of IEEE P1735-based with RTL-based functional+scan locking can thwart malicious actions by insider threats, other oracle-less (OL) and oracle-guided (OG) attacks on logic locking are still applicable. The following justifies how ILP-based RTL-based locking will be against these attacks.

Against OG SAT Attack: Proper implementation of *RTLock* makes the SAT attack impractical (the SAT attack requires full scan access), as step 7 will enable the scan insertion+locking. Nonetheless, the *RTLock* maximizes the SAT resilience based on the ILP-based locking. In the results section, we also reported the SAT attack time while it is not enabled.

Against OG BMC-based Attack: *RTLock* has been built in a way that any algorithmic attack can be used for building the candidate database, and ILP-based candidate selection can efficiently choose the best options based on the targeted attack procedure. In this case, locking candidates are selected in a way that provides deeper resiliency against the BMC attacks. For instance, since unrolling (depth of attack) plays an important role in the scalability of the BMC attack, in FSM locking, deeper states vs. shallow states are selected that enforce the BMC to run for much more clock cycles for executing the attack. Another example is dominant operations in arithmetic-wise candidates. Once an operation is dominant, it will be executed after a set of operations (sequentially), where BMC requires more iterations to retrieve the satisfiability assignments.

Against OL Removal Attack: Signal Probability skew (SPS) based removal attacks [12] depend on the existence of a point function in the design and low output corruptibility. The attack identifies and remove this point function, thus rendering the keys useless. RTLock achieves high corruptibility by introducing keys at the RT level. Also, RTLock does not introduce any point functions in the design, thus foiling the removal attack.

Against OL ML-based Attack: The main limitation of gate-level locking techniques considering the structural attacks is that the key gates are introduced at the already optimized design [18], [37]. Introducing locking gates at the gate level can deviate the design from optimization minima, which are used by ML-based attacks to discern the values of the unlocking key bits. The introduction of keys at RTL by *RTLock* ensures that the design optimizes uniformly in the synthesis stage, making it harder for ML-based attacks to be successful. We show the ML-based attack results in Section V, where the keys were

extracted with high accuracy in the case of gate-level locking techniques, but with RTLock the accuracy fell significantly.

Against OG Bypass Attack: The bypass attack [13] utilizes the low corruptibility feature of the SAT-resistant locking techniques and creates a bypass circuitry to flip the output for the protected input patterns by the locking. For point function-based locking such as SARLock and Anti-SAT [5], [6], a bypass circuit for a single input pattern is sufficient to unlock the design. But RTLock ensures high output corruptibility in the design. To account for the incorrect outputs, the size of the bypass circuitry makes the bypass attack on RTLock infeasible.

Against OG SMT Attacks: The SMT-based attack extracts the behavioral traces on HLS-generated RTL, allowing to recover the locking key [28]. This attack relies on finite state machines with the datapath (FSMD) structure of HLS-based RTLs for the attack purpose. Therefore, it can be highly efficient once datapath and FSM are fully separated in the RTL [22]. However, for the designs implemented in RTL, the datapath and FSM are not necessarily decoupled. The FSM-based locking in *RTLock*, particularly methods like incorrect state transition, creates undesirable state sequences. Each key value creates an incorrect RTL-FSMD (trace) in this case. For n locked transitions, 2^n traces will be built, which must be checked by the SMT attack. Additionally, *RTLock* targets arithmetic operations on which the SMT and KLEE used in the SMT attack are not efficient enough against operations like multiplication and shift, creating hard instances for the SMT (exponential attack time).

V. Results and Analysis

We have tested the proposed *RTLock* framework on a set of HDL benchmark designs, listed in Table II, ranging from small to moderate sizes. These benchmark selections constrain the key size of the locking techniques to unbiasedly realize the resilience against SAT attack. The attack is done by using the SAT tool developed by P. Subramanyan [38]. For evaluation purposes (running post-synth attacks), all the designs are synthesized using *NanGate_15nm* technology library using the Synopsys DC compiler. To comparatively evaluate the resiliency of *RTLock* against different attacks, we consider multiple locking techniques, including RND and MUX2 [3], SLL [31], TOC$_{MUX/XOR}$ [39], and IOLTS [40]. All the experiments are carried out on an Intel Xeon E-2224 32-core processor and 32GB RAM.

TABLE II: The Main Specifications of the Benchmark Circuits.

Circuits	#PI/PO	#Gate	#FFs	Keys	**Circuits**	#PI/PO	#Gate	#FFs	Keys
b05	3/36	1030	34	19	**b15**	38/70	9029	416	38
b14	34/54	10325	215	38	**SHA1**	516/162	10979	849	31
Fibo.	10/91	3449	287	24	**AES**	390/130	26720	2332	45

Table III shows the comparative analysis of the SAT attack [4] when timeout is set to 12h. For comparison purposes, the area overhead across benchmarks for all techniques has been kept at 15%. From the results, it is evident that the proposed *RTLock* provides SAT resilience from 10^2-10^5 for significantly smaller key sizes. This improved resiliency can be associated with creating cases based on the ILP-based analysis of a wide range of candidates suitable to the locking point. Note that increasing the key size, with less impact on the overhead, boosts the robustness achieved by ILP. For instance, for all cases shown in Table III, with doubled key size, the SAT cannot break AES, SHA1, b14/b15 within the timeout. Also, The scan locking at the later stage adds one more PSPACE dimension to the problem size. Due to low scalability, with the same key size, none of the circuits can be broken using the BMC attacks.

TABLE III: SAT Time for Locking Techniques at SAME (15%) Area Overhead.

Method	Benchmark Circuits											
	AES		SHA1		b14		b15		Fibo.		b05	
	‖k‖	t	‖k‖	t	‖k‖	t	‖k‖	t	‖k‖	t	‖k‖	t
RND [3]	498	8.2	360	3.13	310	5.6	302	6.7	247	3.7	151	2.1
SLL [31]	562	181.2	492	242	417	119	425	126	326	97	182	23
TOC$_{MUX}$ [39]	352	1.8	283	2.3	251	2.2	247	2.7	213	2.1	131	1.1
TOC$_{XOR}$ [39]	287	16.9	244	13.3	213	8.2	227	9.7	207	8.1	112	5.5
IOLTS [40]	986	3.1	793	0.92	632	1.1	641	1.4	473	1.2	254	1.1
RTLock*	**35**	**36350**	**25**	**16758**	**32**	**4286**	**32**	**9619**	**16**	**2094**	**16**	**911**

*:RTLock without scan locking ‖k‖: key size ‖t‖: attack time (second)

TABLE IV: ML-based Attack [18], [37] Accuracy on Locking.

Circuits	ML Accuracy (%) on Locking Solutions							
	TOC$_{MUX}$ [39]		IOLTS [40]		MUX2 [3]		**RTLock***	
	‖key‖	Acc%	‖key‖	Acc%	‖key‖	Acc%	‖key‖	Acc%
AES	352	{97.4, 96.6}*	986	{99.1, 99.4}	373	{94.1, 93.6}	35	{54.3, 51.4}
SHA1	283	{97.5, 96.8}	793	{100, 100}	298	{92.7, 94.3}	25	{48, 48}
b14	251	{97.6, 97.2}	632	{100, 99.4}	256	{93.8, 93.4}	32	{50, 46.9}
b15	247	{97.6, 97.2}	641	{99.4, 99.3}	257	{93.8, 93.7}	32	{59.4, 53.1}
Fibo.	213	{96.2, 97.2}	473	{100, 99.1}	223	{94.2, 93.3}	16	{56.3, 50}
b05	131	{96.9, 97.7}	254	{98.8, 99.2}	136	{92.6, 93.4}	16	{50, 56.2}
AVG	**246**	**{97.2, 97.1}**	**630**	**{99.6, 99.5}**	**257**	**{93.5, 93.6}**	**26**	**{52.9, 50.9}**

*:RTLock without scan locking.
*: {x1, x2}, where x1 is SWEEP accuracy and x2 is SCOPE accuracy.

To show the robustness against ML-based attacks, we performed SWEEP and SCOPE frameworks on locked circuits [18], [37]. Table IV shows the accuracy (ratio of correctly predicted keys). It is very high for the gate-level techniques, while it is much lower in *RTLock* ($\sim$ 50%), and this low accuracy is due to the uniform distribution of operators while creating and inserting the locking cases[1].

TABLE V: Testability Coverage: Regular ATPG vs. Locking-aware ATPG.

Circuits	**One** Key Constraint Set			**Multiple** Key Constraint Sets			
	Test Coverage	Fault Coverage	# of patterns	Test Coverage	Fault Coverage	# of Patterns	# of key Constraints
AES	99.97%	96.21%	705	99.99%	99.25%	274	2
SHA1	99.24%	96.63%	356	99.91%	99.88%	193	3
Fibo.	99.80%	96.83%	251	99.97%	97.87%	183	2
b05	99.34%	92.72%	68	99.74%	93.4%	59	2
b14	99.83%	98.51%	1081	99.65%	98.14%	1203	4
b15	99.25%	98.61%	628	99.21%	98.59%	638	3

Table V reflects the testability results of the *RTLock*ed circuits. Two sets of testability results are reported in Table V by applying (i) one key constraint and (ii) multiple key constraints to the key inputs. While generating the test patterns for one key constraint, we utilized the post-test activation method [41] by applying one dummy key value to the key inputs. Table V shows the test coverage, fault coverage, and the number of test patterns. Even though *RTLock* performed partial scan insertion and scan locking at the RTL, test coverage is $> 99\%$ for all cases. Multiple key constraints [42] make it easier for the ATPG tool to detect the undetectable (or difficult to detect) faults, thus decreasing the number of test patterns and increasing test coverage in most cases. This shows that *RTLock* keeps manufacturing yield intact.

Table VI shows the post-layout PPA overhead in two different modes: (i) only functional locking, (ii) functional+scan

[1]ML-based Maximum resiliency is when the results produce 50% accuracy. When it is close to 0% the weight metrics can be tuned to gain a high accuracy near 100%.

TABLE VI: PPA (Post-layout) Overhead of the RTLock Locked Circuits.

Circuit	Original			Functional*1			Functional+Scan*2		
	Area (um^2)	Delay (ns)	Power (mW)	Area (%)	Delay (%)	Power (%)	Area (%)	Delay (%)	Power (%)
AES	62938.76	0.87	3.06	8.66	7.03	0	9.81	3.83	0
SHA1	21314.55	0.30	0.77	13.80	11.61	3.90	13.45	7.18	2.60
Fibo.	7297.24	0.08	1.03	14.28	11.71	0.80	35.02	4.80	5.30
b05	1671.25	0.02	5.52	23.75	18.26	4.70	9.06	14.23	-0.30
b14	23685.20	0.28	0.13	25.24	31.54	-0.10	30.14	19.80	0.80
b15	18034.06	0.23	2.04	23.86	25.17	5.50	21.80	14.20	4.80

*1: Overhead of Purely RTL Functional Locking using *RTLock*.
*2: Overhead of RTL Functional + RTL Scan Locking using *RTLock*.

locking. We chose post-layout to consider the impact of RTL manual scan insertion. Please note that the PPA overhead corresponding to functional locking is normalized based on the original design, whereas the PPA overhead of functional+scan locking is based on the functional part. This helps to clarify the impact of RTL scan locking. As shown, *RTLock* incurs reasonable overhead while the circuit size is moderate to large, e.g., AES, where overhead is less than 10%.

VI. Conclusion

RTLock is a robust RTL-based locking technique that considers each entity of the IC design process untrusted. *RTLock* selects the locking point based on an ILP-based analysis that relies on comprehensive pre-computed single-at-once locking candidates w.r.t. design logic and operations. Further, *RTLock* enables the possibility of scan obfuscation at RTL to push all locking-oriented actions toward the earliest stage of the design. Based on the refined threat model, we showed why RTL-based locking must be equipped with P1735 to be protected against insider threats. Our experiments show that *RTLock* is effective against a wide threat model with high resilience at low overhead without compromising test coverage.

Acknowledgement

We would like to acknowledge Dr. M. Sazadur Rahman for his diligent work and valuable input throughout the paper.

References

[1] M. Rostami *et al.*, "A Primer on Hardware Security: Models, Methods, and Metrics," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.
[2] H. M. Kamali *et al.*, "Advances in Logic Locking: Past, Present, and Prospects," *Cryptology ePrint Archive, 2022/260*, 2022.
[3] J. Roy *et al.*, "Ending Piracy of Integrated Circuits," *Computer*, vol. 43, no. 10, pp. 30–38, 2010.
[4] P. Subramanyan *et al.*, "Evaluating the security of logic encryption algorithms," in *Hardware Oriented Security and Trust Symposium (HOST)*, 2015, pp. 137–143.
[5] M. Yasin *et al.*, "SARLock: SAT attack Resistant Logic Locking," in *Hardware Oriented Security and Trust Symposium (HOST)*, 2016, pp. 236–241.
[6] Y. Xie and A. Srivastava, "Mitigating SAT Attack on Logic Locking," in *CHES*, 2016, pp. 127–146.
[7] M. Yasin *et al.*, "Provably-Secure Logic Locking: From Theory to Practice," in *ACM SIGSAC Conference on CCS*, 2017, pp. 1601–1618.
[8] H. M. Kamali *et al.*, "Lut-lock: A novel LUT-based Logic Obfuscation for FPGA-bitstream and ASIC-hardware Protection," in *IEEE Computer Society Symp. on VLSI (ISVLSI)*, 2018, pp. 405–410.
[9] S. Roshanisefat *et al.*, "SRCLock: SAT-resistant Cyclic Logic Locking for Protecting the Hardware," in *GLSVLSI*, 2018, pp. 153–158.
[10] H. M. Kamali *et al.*, "Full-lock: Hard Distributions of SAT Instances for Obfuscating Circuits using Fully Configurable Logic and Routing Blocks," in *DAC*, 2019, pp. 1–6.
[11] M. S. Rahman *et al.*, "Security Assessment of Dynamically Obfuscated Scan Chain Against Oracle-guided Attacks," *ACM TODAES*, vol. 26, no. 4, pp. 1–27, 2021.
[12] M. Yasin *et al.*, "Removal attacks on logic locking and camouflaging techniques," *IEEE Transactions on Emerging Topics in Computing*, vol. 8, no. 2, pp. 517–532, 2017.
[13] X. Xu *et al.*, "Novel Bypass Attack and BDD-based Tradeoff Analysis against All Known Logic Locking Attacks," in *CHES*, 2017, pp. 189–210.
[14] K. Shamsi *et al.*, "AppSAT: Approximately Deobfuscating Integrated Circuits," in *Hardware Oriented Security and Trust Symposium (HOST)*, 2017, pp. 95–100.
[15] H. Zhou *et al.*, "CycSAT: SAT-based Attack on Cyclic Logic Encryptions," in *Int'l Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 49–56.
[16] P. Chakraborty *et al.*, "SAIL: Machine Learning Guided Structural Analysis Attack on Hardware Obfuscation," in *AsianHOST*, 2018, pp. 56–61.
[17] K. Z. Azar *et al.*, "SMT Attack: Next Generation Attack on Obfuscated Circuits with Capabilities and Performance Beyond the SAT Attacks," *IACR TCHES*, 2019.
[18] A. Alaql *et al.*, "Sweep to the secret: A constant propagation attack on logic locking," in *AsianHOST*, 2019, pp. 1–6.
[19] K. Z. Azar *et al.*, "NNgSAT: Neural Network guided SAT Attack on Logic Locked Complex Structures," in *Int'l Conf. On Comp. Aided Design (ICCAD)*, 2020, pp. 1–9.
[20] S. Roshanisefat *et al.*, "RANE: An Open-Source Formal De-obfuscation Attack for Reverse Engineering of Logic Encrypted Circuits," in *GLSVLSI*, 2021, pp. 221–228.
[21] C. Pilato *et al.*, "TAO: Techniques for Algorithm-level Obfuscation during High-Level Synthesis," in *DAC*, 2018, pp. 1–6.
[22] C. Karfa, *et al.*, "HOST: HLS Obfuscations against SMT ATtack," in *Design, Automation & Test in Europe Conference (DATE)*, 2021, pp. 32–37.
[23] M. R. Muttaki *et al.*, "HLock: Locking IPs at the High-Level Language," in *DAC*, 2021, pp. 79–84.
[24] Md. R. Muttaki *et al.*, "Hlock+: A robust and low-overhead logic locking at the high-level language," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.
[25] C. Pilato *et al.*, "ASSURE: RTL locking against an Untrusted Foundry," *IEEE Transactions on VLSI Systems*, vol. 29, no. 7, pp. 1306–1318, 2021.
[26] N. Limaye *et al.*, "Fortifying RTL Locking Against Oracle-Less (Untrusted Foundry) and Oracle-Guided Attacks," in *DAC*, 2021, pp. 91–96.
[27] D. Sisejkovic *et al.*, "Designing ML-Resilient Locking at Register-Transfer Level," 2022.
[28] C. Karfa *et al.*, "Is Register Transfer Level Locking Secure?" in *Design, Automation & Test in Europe Conference (DATE)*, 2020, pp. 550–555.
[29] WG for Design IP Encryption and Rights Management, "Recommended Practice for Encryption and Management of Electronic Design Intellectual Property (IP)," in *IEEE Std 1735-2014*, 2015, pp. 1–90.
[30] D. Sisejkovic *et al.*, "Challenging the Security of Logic Locking Schemes in the era of Deep Learning: A Neuroevolutionary Approach," *ACM JETC*, 2021.
[31] J. Rajendran *et al.*, "Security analysis of logic obfuscation," in *DAC*, 2012, pp. 83–89.
[32] R. Kibria *et al.*, "Fsmx: Finite state machine extraction from flattened netlist with application to security," in *2022 IEEE 40th VLSI Test Symposium (VTS)*, 2022, pp. 1–7.
[33] A. Schrijver, *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998.
[34] L. Goldstein and E. Thigpen, "Scoap: Sandia controllability/observability analysis program," in *Design Automation Conference*, 1980, pp. 190–196.
[35] Libero SoC Secure IP Flow User Guide, "http://www.microsemi.com/document-portal/docview/133573-libero-soc-secure-ip-flow-user-guide."
[36] P. Rogaway, "Authenticated-encryption with associated-data," in *ACM Conference on CCS*, 2002, pp. 98–107.
[37] A. Alaql *et al.*, "SCOPE: Synthesis-based constant propagation attack on logic locking," *IEEE Transactions on VLSI Systems*, vol. 29, no. 8, pp. 1529–1542, 2021.
[38] P. Subramanyan, "https://git.uwaterloo.ca/jmshahen/LogicLocking-Empirical/-/tree/master/SAT/spramod-host15-logic-encryption-90a8ca47d847."
[39] J. Rajendran *et al.*, "Fault analysis-based logic encryption," *IEEE Transactions on Computers*, vol. 64, no. 2, pp. 410–424, 2013.
[40] S. Dupuis *et al.*, "A Novel Hardware Logic Encryption Technique for Thwarting Illegal Overproduction and Hardware Trojans," in *IOLTS*, 2014, pp. 49–54.
[41] M. Yasin *et al.*, "Activation of logic encrypted chips: Pre-test or post-test?" in *Design, Automation & Test in Europe Conference (DATE)*, 2016, pp. 139–144.
[42] M. S. Rahman *et al.*, "LL-ATPG: Logic-Locking Aware Test Using Valet Keys in an Untrusted Environment," in *IEEE ITC*, 2021, pp. 180–189.