Fortifying RTL Locking Against Oracle-Less (Untrusted Foundry) and Oracle-Guided Attacks

Nimisha Limaye*, Animesh B. Chowdhury*, Christian Pilato†, Mohammed T. M. Nabeel‡,
Ozgur Sinanoglu‡, Siddharth Garg*, Ramesh Karri*
*New York University, USA, †Politecnico di Milano, Italy, ‡New York University Abu Dhabi, UAE

***Abstract*—Logic locking protects integrated circuits (IC) against intellectual property (IP) theft, IC overbuilding, and hardware Trojan insertion. Prior locking schemes operate after logic synthesis and cannot protect the semantic information embedded into the logic. Register-transfer level (RTL) locking can protect the sensitive IP semantics and are EDA tool-chain agnostic, allowing seamless integration into arbitrary design flows. State-of-the-art RTL locking protects against the untrusted foundry assuming no access to working chip (oracle). However, it does not protect against oracle-based attacks. In this work, we propose to fortify RTL locking to protect against *all* untrusted entities in the supply chain, including foundry for oracle-less attacks, and test facility and end users for oracle-guided attacks.**

I. Introduction

Increasing fabrication costs due to technology scaling created a globally distributed integrated circuit (IC) supply chain wherein IC fabrication, testing, and packaging are outsourced to different entities around the globe to amortize this cost [1]. The presence of potentially untrustworthy entities in the IC supply chain opened up avenues for attackers to pirate confidential intellectual property (IP), overproduce ICs, and to insert malicious hardware Trojans. An attacker with access to a reverse-engineered netlist obtained from the GDSII at the foundry or by physical reverse engineering as an end user could get the IP information and either overbuild it or steal it.

Logic locking seeks to safeguard designs from such attacks [2]. In the past, locking schemes inserted key gates either in the gate-level netlists or structural register-transfer level (RTL) files [3], [4]. Obfuscation was delegated to security-oblivious and deterministic synthesis tools; this opened the doors to structural-analysis attacks [5], [6]. When an attacker has access to the functional IC (the *oracle*), oracle-guided attacks like Boolean satisfiability (SAT)-based ones can be launched to retrieve the secret locking key [7]. Further, compound locking scheme [8] proposed to thwart oracle-guided and structural attacks at once is also shown to be broken [9].

Researchers proposed high-level logic locking schemes during high-level synthesis (HLS) [10] or directly on behavioral RTL [11]. As shown in Fig. 1, these schemes operate on a technology-independent representation of the design, making the obfuscated designs portable and reusable across different ICs. However, these schemes currently focus on thwarting an untrusted foundry (oracle-less threat model) and cannot defend against oracle-guided attacks [7], [12] (see Table I). For widespread impact, these approaches must be extended to protect the IPs against attacks by all entities in the IC supply chain, with or without an oracle.

The research is supported in part by NSF (A#: 1526405), DARPA Automatic Implementation of Secure Silicon (AISS) program (#: HR0011-20-9-0043), the NYU Center for Cybersecurity (cyber.nyu.edu), and NYUAD Center for Cybersecurity (sites.nyuad.nyu.edu/ccs-ad).

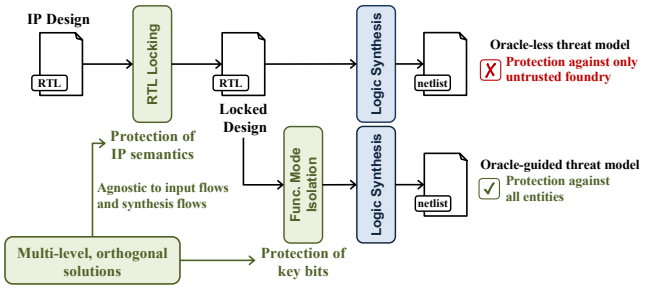


Fig. 1: Designer can protect semantically meaningful, sensitive information by operating at the register-transfer level (RTL). This can thwart an untrusted foundry that has no access to an oracle. Isolating functional mode from the test mode further protects against all other entities that have access to an oracle.

TABLE I: Comparison with prior work

	Attack	Untrusted foundry	Untrusted end user
TAO [10]	SMT [12]	✓	✗
ASSURE [11]	SAT [7]	✓	✗
This work	-	✓	✓

In this work, we fortify existing RTL locking scheme [11] against oracle-based attacks by combining it with functional mode isolation (FMI) methodology. The oracle-based attacks succeed in recovering the secret key mainly when they have access to both the internal system state and the primary outputs. FMI methodology disables access to the internal system state when it senses activity on the scan infrastructure. This combined defense achieves the required security and output corruption to thwart all the untrusted entities in the supply chain. Composing the orthogonal solutions does not affect the complementary security guarantees that each offers. Instead, it provides a multi-layer protection, where RTL locking hides the semantics from the foundry and FMI protects against oracle-based attacks. **The four key contributions of this work are:**

1) While RTL locking claims proven resilience to oracle-less attacks, for the first time we assess if this is indeed the case. We show that RTL locking thwarts oracle-less redundancy attacks [5] and explain where it applies.
2) We add RTL functional mode isolation (FMI) as an

additional security layer to thwart oracle-guided attacks by design (e.g., SAT attacks [7]) that rely on scan access.
3) Using the oracle guided attack tool KC2 [13], we show that FMI thwarts such attacks and clarify where it applies.
4) We demonstrate the scalability of integrated RTL locking on several RTL benchmarks [14] and on the tape-out-ready Cortex-M0 micro-controller targeting *GlobalFoundries 55nm LPe*.

II. Background and Related Work

A. Threat Model

There are two broad categories of threat models, viz., oracle-less and oracle-guided. Under oracle-less threat model, we assume the attacker is an untrusted foundry that has access only to a reverse-engineered netlist from which he/she can extract information about operations and signals. Under the oracle-guided threat model, we assume the attacker is an untrusted end user and has access to the functional IC (oracle) and can obtain the underlying netlist by either colluding with an untrusted foundry or reverse-engineering the IC.

Prior RTL locking solutions [10], [11] focus on the oracle-less threat model for low-volume IC production. In this work, we aim at fortifying this approach to consider also an oracle-guided threat model where the **attacker has access to both the reverse-engineered netlist and a working IC (oracle)**. In this case, we assume the attacker to be at any point in the IC supply chain: an untrusted foundry, test facility, or end user.

B. Logic Locking

Logic locking is a design-for-trust solution to safeguard ICs against hardware attacks. It inserts key gates driven by secret key inputs later stored in a non-volatile tamper-proof memory located on the IC. When the secret key is applied, the IC becomes functional and provides correct results. For all incorrect keys, the IC produces erroneous outputs. Logic locking was conventionally performed either on structural RTL designs or technology mapped gate-level netlists. Recent works proposed to operate at the behavioral RTL (before logic synthesis) to obfuscate the behavior instead of the structure.

C. RTL Locking Against Oracle-Less Attacks

Logic synthesis absorbs vital security semantics and generates an optimized netlist. Gate-level locking takes in a netlist as input and inserts key-input controlled XOR/multiplexer logic gates in it. Logic-locked gate-level optimized designs are prone to oracle-less de-synthesis [15] and structural [16], [6] attacks. RTL locking obfuscates a design at a higher-level of abstraction than the common gate-level approaches. RTL obfuscation uses the semantic information available at the behavioral level to create a set of totally valid alternatives. Since the RTL is locked before the synthesis tools can apply logic optimizations, it prevents vulnerabilities induced by the IC structure. High-level locking techniques such as TAO [10] and ASSURE [11] borrow concepts from software obfuscation like obfuscating control-flow graphs, data structures, and operations in the program. TAO [10] does obfuscation during high-level synthesis (HLS) requiring access to the internals of the HLS tool. ASSURE [11], instead, obfuscates the RTL designs and hence is oblivious to the source of the RTL. While these approaches are simple and easy to use, they are not designed to be resilient to oracle-guided attacks [7], [12], limiting the applicability to commercial, high-volume IC designs.

III. Fortifying RTL Locking Against Oracle-less and Oracle-based Attacks

As shown in Fig. 1, our approach is composed of two subsequent steps: it first applies RTL locking to protect the IC semantics against reverse engineering and then functional mode isolation to protect the key against oracle-based attacks.

A. Protecting IC Design by Obfuscating Behavioral Semantics

Gate-level netlist locking or structural RTL locking leaves behind traces from deterministic synthesis steps which are then leveraged by structural/netlist-analysis-based attacks to recover the secret locking key [5], [6], [15], [16]. In our work, we lock the behavioral RTL using a **behavioral semantics obfuscation** (BSO) to hide the IC semantics instead of its structure; we use ASSURE [11] to perform BSO. ASSURE offers full control to the designer to select which secret and confidential constants (like proprietary coefficients), arithmetic operations, and control branches to lock. Using this approach, we replace these elements with predicates that are dependent on the external locking key; each element is associated with a specific part of the key. Only when applying the correct key, the design operates on the original constants, operations, and branches. This embeds a `100%` error rate in the design for even one-bit change in the key configuration (i.e., no collision). We now briefly describe the BSO techniques implemented in ASSURE for locking a given design.

1) **Constant obfuscation**: We protect confidential constants by replacing them entirely with key bits carrying constant information. For example, we rewrite the operation $out = in + 8'b11101001$ as $out = in + K_c$ where K_c is the 8-bit constant stored in the locking key. The attacker has 2^8 possibilities to get a correct out for a given in value.
2) **Operation obfuscation**: We introduce a multiplexer to decide between the correct operation and a dummy operation based on the value of one key bit. For example, we lock the operation $out = in_1 + in_2$ as $out = K_o$? (in_1 + in_2) : (in_1 - in_2). Only the correct value of K_o enables the propagation of the correct result.
3) **Branch obfuscation**: In branch obfuscation, we perform XOR based locking on the original condition followed by logical inversion if key-bit is 1. For e.g., we rewrite the original condition $in_1 > in_2$ as $(in_1 \leq in_2) \oplus K_b$ with locking key-bit $K_b = 1$. An attacker cannot deduce from the statement whether the original condition was $>$ or $\leq$.

The tool outputs a behavioral, locked RTL that adds an extra input port to the interface to load the locking key. The locked

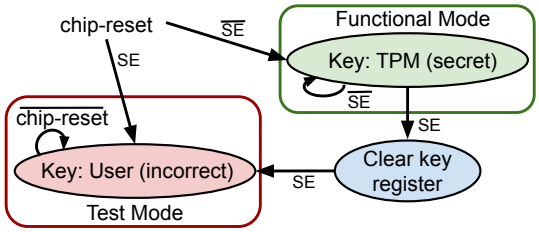


Fig. 2: Isolation of functional mode explained using a finite state machine (FSM). If activity is sensed on the scan-enable (SE) signal, the circuit moves out of the functional mode.

TABLE II: Functional mode isolation (FMI) operation

Reset	SE	Key-register
1	x	FMI reset
$1 \rightarrow 0$	1	Key from scan-in
$1 \rightarrow 0$	0	Key from TPM
0	$0 \rightarrow 1$	1. FMI reset 2. Key from scan-in

RTL design can be taken through the same synthesis flow used for the original, non-secure design.

B. Functional Mode Isolation

RTL locking hides the sensitive semantic information in the original design thereby thwarting structural-analysis (oracle-less) attacks (see Table I). But when attackers have access to a working IC (oracle), they can query the oracle through its scan infrastructure and primary outputs to recover the secret information. Obfuscation and oracle-guided protection are orthogonal and can be applied at the same/different abstraction levels. To undermine attackers with access to a reverse-engineered netlist and an oracle, we provide an RTL solution for **functional mode isolation** (FMI) to protect the key upon sensing scan access.

FMI adds a *key register* that drives the key of the locked RTL design. In functional mode, the key register is loaded with the secret key from the tamper-proof memory immediately after global reset, ensuring correct functionality. When an oracle attack creates an activity on the scan-enable (`SE`) signal, the chip exits the functional mode while the FMI circuitry immediately clears the key register and isolates it from the tamper-proof memory. This process breaks the functionality making the oracle unusable. Test and debug are both allowed but with the key register loaded with dummy user keys through the test interface, similar to the methods used in oracle-less attacks with random key guesses [11]. The correct functionality can only be restored with a global reset. A similar approach was recently proposed in [17]. While DisORC demonstrates resilience against oracle-guided attacks, it is only applied at the gate-level for a locked netlist. Scan locking techniques [18], [19], [20] which insert key-driven logic on scan paths are also shown to be vulnerable to modeling-based attacks [21], [22].

Architecture: Figure 2 depicts the FMI operation as a finite state diagram. On a global `chip-reset`, the circuit starts

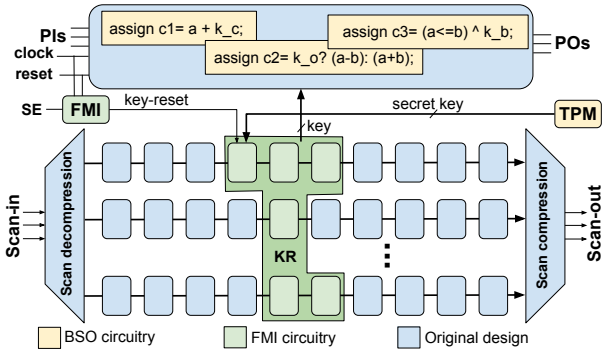


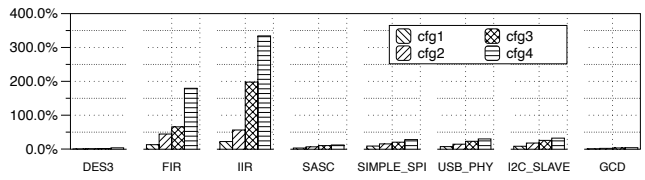
Fig. 3: Integrated BSO + FMI architecture. KR is the key register consisting of key scan cells distributed over multiple scan chains based on synthesis and layout constraints. Key scan cells need not necessarily be placed as shown. SE is the scan-enable signal. TPM is tamper-proof memory.

either in the test or functional mode based on the value of `SE` signal. During functional mode, an activity on the `SE` signal resets the key register. The chip enters (correctly) in test mode but the design receives an incorrect key (i.e., reset values of the key registers). Only with a global reset, the circuit returns to functional mode. We provide a behavioral RTL implementation of the FMI unit by following the operations in Table II. The FMI unit requires a sensing circuitry for the `SE` signal and a reset circuitry for the key register. This unit is independent of the rest of the design and does not require low-level scan architecture details. During scan-chain insertion, the `SE` pin is reused and the key register becomes part of the scan chains. The key scan cells may be distributed across different scan chains based on layout constraints; this has no implication on security as key registers are reset immediately upon scan activity sensing and before any scan shift operation takes place. Figure 3 describes the integration of the BSO and FMI approaches. We also show how the selected locking key bits are registered, integrated, and protected in the scan chain.

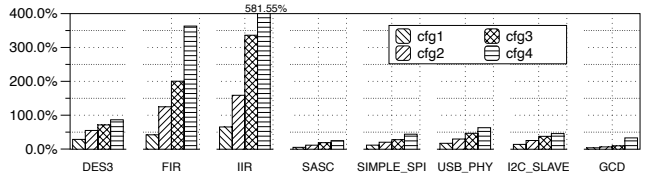
Testing: Scan-chains in a design facilitate efficient structural testing of the ICs. Structural testing can be carried out using an incorrect key as the correct functionality is not required. So, our FMI method supports an untrusted test facility in the supply chain. The FMI circuitry around the RTL locking solution is tested implicitly during structural and functional tests. For in-field debug, a trusted entity can bring the circuit to a known state even in the test mode by providing the correct key through the scan chains.

C. Security Analysis

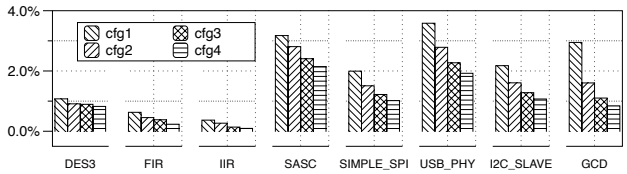
As the two approaches are orthogonal, we analyze the security of each step separately. Our solution protects against an untrusted foundry using indistinguishable obfuscation [11]. BSO produces a behavioral RTL design which any front-end and back-end synthesis tools can use to produce layout files. The synthesis process successfully dissolves the traces of the locking components present in the obfuscated behavioral RTL design, thereby making it structurally indistinguishable from



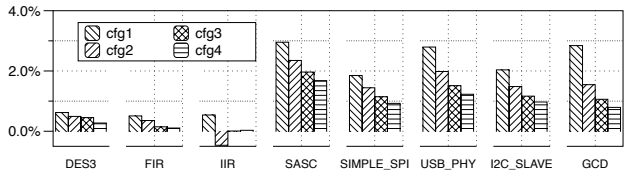
(a) BSO overhead when locking only constants.



(b) BSO overhead when locking all element types.

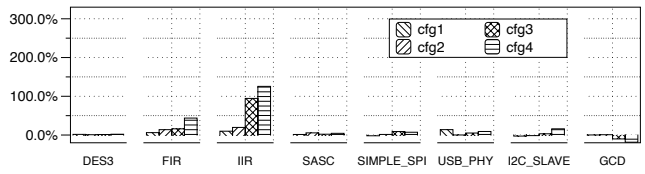


(c) FMI overhead when locking only constants.

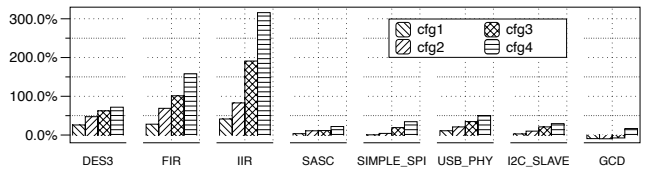


(d) FMI overhead when locking all element types.

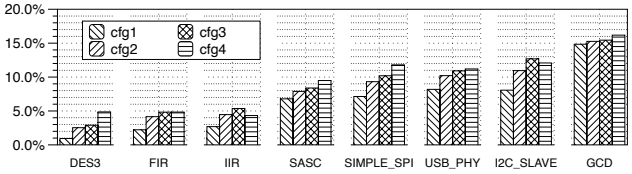
Fig. 4: Area overhead when applying the combination of the proposed schemes. We report the BSO overhead compared to the original design and the FMI overhead compared to the design already protected with BSO.



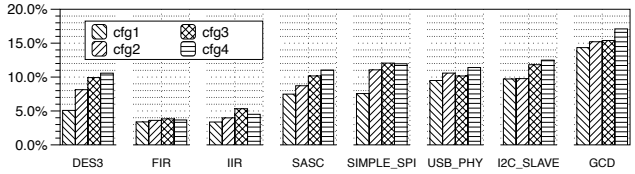
(a) BSO overhead when locking only constants.



(b) BSO overhead when locking all element types.



(c) FMI overhead when locking only constants.



(d) FMI overhead when locking all element types.

Fig. 5: Power overhead when applying the combination of the two proposed schemes. We report the BSO overhead compared to the original design and the FMI overhead compared to the design already protected with BSO.

the original design. However, current RTL locking methods are secure only against oracle-less attacks [10], [12].

FMI is a complementary approach that eliminates access to the scan chain, thwarting all scan-based attacks as variants of SAT-based attacks. While structural-analysis attacks can recover the secret key in this threat model, locking at the RTL thwarts this type of analysis, as shown by the experiments in Section IV. FMI only protects the scan chains; the attacker still has access to the oracle. We do a security analysis using model checkers and show that even for small circuits, the attack fails under resource and time constraints.

IV. Experimental Results

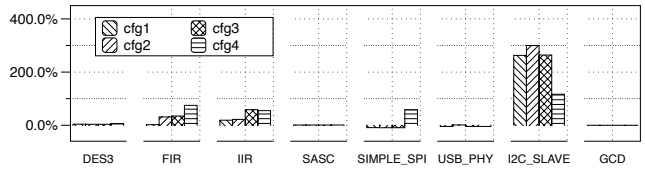
A. Experimental Setup

In our experiments, we used selected designs from *CEP*, *IWLS-05*, *OpenCores*, and *OpenROAD* benchmark suites. We used ASSURE for BSO and an in-house tool for FMI generation. We performed logic synthesis with *Synopsys Design Compiler* targeting the *Nangate 45nm Open Cell Library* [23]. We performed test coverage analysis with *Synopsys Tetramax* and functional verification with *Synopsys VCS*. All these experiments are carried out on *CentOS* 6 operating system with `128`-core Intel Xeon processor running at `2.2` GHz.

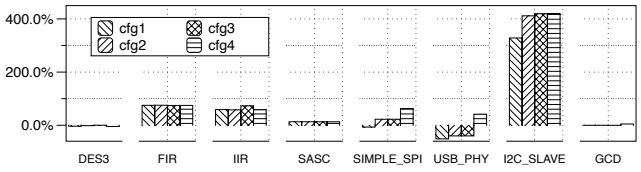
To validate the security promises of our solution, we run the open-source KC2 attack based on model checkers [13] and the redundancy attack based on structural analysis [5]. Due to code dependencies, attack experiments are carried out on *Linux Ubuntu 16.04* operating system with `24`-core Intel Xeon processor running at `3.3` GHz having `96` GB RAM. For KC2 experiments, each run allocates `10,000` MB of memory. For both the attacks, we set a time-out of `48` hours.

B. Overhead Analysis

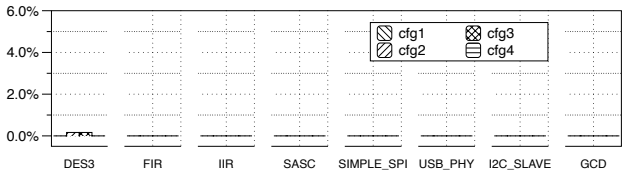
We assess the practicality of our defense using area, power, and timing overhead compared to original designs when varying the designer's requirements. As in [11], we create four configurations for each design, namely CFG1, CFG2, CFG3, CFG4 using an increasing number of key bits. CFG4 always corresponds to using the maximum number of bits. Due to space limitations, we present results when all techniques



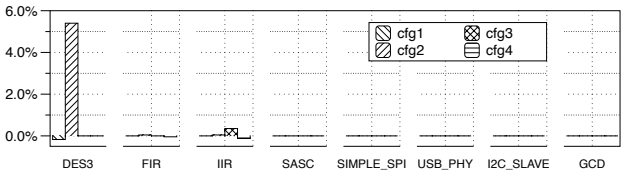
(a) BSO overhead when locking only constants.



(b) BSO overhead when locking all element types.



(c) FMI overhead when locking only constants.



(d) FMI overhead when locking all element types.

Fig. 6: Timing overhead (critical-path delay) when applying the combination of the two proposed schemes. We report BSO overhead compared to the original design and the FMI overhead compared to the design already protected with BSO.

and only constants are used for locking, while all others are available to the community [14].

Figure 4 shows area overhead. On average, we incur 59.36% area overhead for BSO and an additional 1.29% overhead due to FMI. The area overhead is larger for data-intensive designs and grows with the size of the key. The size of the FMI circuitry is, instead, only proportional to the number of registered key bits and not the size of the design. So, it becomes less noticeable in large locked designs.

Figure 5 shows power overhead. On average, we incur 28.91% power overhead when applying the BSO scheme and an additional 8.66% overhead for the FMI circuitry. While BSO introduces extra gates (additional static power), the dynamic activity is related only to the original signals. So, power overhead follows the area trend. Extra activity (and overhead) is, however, required for the FMI circuitry.

Figure 6 shows timing overhead. While no extra cycles are added to the circuit, the extra logic may affect the critical paths. On average, we incur 54.17% performance overhead corresponding to the BSO scheme and 0.09% overhead corresponding to the FMI circuitry. Figure 7 presents test coverage for original and RTL-locked design; we incur on average a loss of 0.1% when designs are locked with BSO.

C. Security Analysis

We analyze the security of our locking scheme using open-source oracle-less and oracle-guided attacks.

Oracle-less attack (Redundancy [5]): The attack exploits redundancies introduced by synthesis with incorrect keys. It works on the principle of testing; with correct key, unlocked and original designs should have the same fault coverage. Incorrect keys might introduce some redundancies, reducing the fault coverage. This analysis aides the redundancy attack in deciphering the correct key.

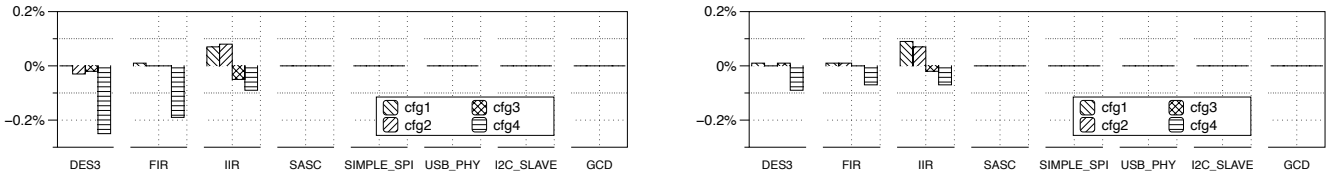
Oracle-guided attack (KC2 [13]): KC2 improves prior model-checker based attacks, which unroll the sequential circuit and recover the key using only primary inputs and outputs; it works with no scan-chain access.

TABLE III: Security assessment of our defense against oracle-less redundancy attack [5] and oracle-guided KC2 attack [13]. *Failed* denotes attack failure due to non-existence of untestable faults for redundancy attack and due to unsolvable constraints for KC2 attack. *Timeout* denotes tool termination after 48 hours without returning the key.

Design (max bits)	Cfg	Used key bits	Oracle-less [5] Key-bits recovered	Oracle-less [5] Exe. time (sec)	Oracle-guided [13] Key-bits recovered	Oracle-guided [13] Exe. time (sec)
DES3 (898)	CFG1	203	2/3	1,984	203	95,588
	CFG2	448	0/56	6,224	0	Timeout
	CFG3	647	12/75	12,334	0	Timeout
	CFG4	898	6/119	29,990	0	Timeout
FIR (344)	CFG1	70	0	Failed	70	308
	CFG2	172	0	Failed	172	2,740
	CFG3	242	0	Failed	239	19,926
	CFG4	344	0	Failed	0	Timeout
GCD (31)	CFG1	8	1/5	2	8	3,288
	CFG2	16	3/10	2	16	2,218
	CFG3	24	4/14	3	24	2,965
	CFG4	31	7/18	4	31	2,827
IIR (651)	CFG1	139	0	Failed	139	4,825
	CFG2	310	0	Failed	310	58,602
	CFG3	480	0	Failed	0	Timeout
	CFG4	651	0	Failed	0	Timeout
SASC (126)	CFG1	32	0	Failed	0	Timeout
	CFG2	64	0	Failed	0	Timeout
	CFG3	94	0	Failed	0	Timeout
	CFG4	126	0	Failed	0	Timeout
SPI (288)	CFG1	73	0	Failed	0	Failed
	CFG2	145	0	Failed	0	Failed
	CFG3	216	0	Failed	0	Failed
	CFG4	288	0	Failed	0	Failed
USB_PHY (223)	CFG1	57	9/27	18	0	Timeout
	CFG2	112	0	Failed	0	Timeout
	CFG3	168	48/97	38	0	Timeout
	CFG4	223	48/115	79	0	Timeout

For redundancy attack: X/Y means out of Y key-bits claimed to be recovered by the attack, X key-bit values are correct.

The oracle-less redundancy attack and the oracle-guided KC2 attack are excellent candidates to validate the security of our proposed solution. Table III provides the attack results



(a) Locking only constants.

(b) Locking all element types.

Fig. 7: Test coverage (TC) overhead. On average, there is only a 0.1% reduction in TC when BSO is integrated with FMI.

TABLE IV: Overhead analysis of Cortex-M0 when locked with BSO and FMI using a 128-bit key. TC: Test coverage.

Cortex-M0	Area (μm^2)	Power (mW)	Critical Path (ns)	TC (%)
Original	46,748	3.26	4.19	100
BSO (keyreg)	60,074	4.22	4.39	100
BSO+FMI (keyreg)	60,811	4.25	4.44	100

when all elements are locked.[1] Combining BSO+FMI protects the designs from both oracle-less and oracle-guided attacks.

We identified cases (**DES**, **FIR**, and **USB_PHY**) where redundancy attacks can recover a few bits from locked designs, showing vulnerabilities in the tool implementation. Integration with FMI exposes some vulnerabilities in oracle-guided attacks (see **FIR** and **GCD**, and some configurations of **DES3** and **IIR**). These results can help us improve locking methods.

D. Case Study

Table IV presents the overhead for ARM Cortex-M0 microcontroller when locked using our solution and synthesized using the tape-out flow with *GlobalFoundries 65nm LPe* technology to showcase the practicality of a silicon-grade implementation. Timing of the design is closed at 250 MHz across the corners. Synthesis is done using *Synopsys Design Compiler*. *Synopsys IC Compiler* is used for physical design. Timing and power analysis are done using *Synopsys Prime Time* and *Prime Power*, respectively. Functional simulation is performed using *Synopsys VCS*. All EDA tools are executed on a 128-core Intel Xeon processor running at 2.2 GHz. We performed BSO with a 128-bit key. From Table IV, we observe 28.51%, 29.45%, and 4.77% overhead for area, power, and critical path, respectively. The integration with FMI introduces 1.23%, 0.71%, and 1.14% overhead for area, power, and critical path, respectively. Further, as seen from Table IV, test coverage is unchanged.

V. Conclusion

RTL locking proved to successfully thwart oracle-less attacks utilizing only structural information of the underlying netlist [10], [11]. However, these solutions were broken by satisfiability modulo theories (SMT) [12] and SAT-based attacks when an oracle is available. We fortify RTL locking by functional mode isolation to make a complete high-level solution for achieving end-to-end protection against all untrusted entities in the IC supply chain. We give the control of area, power, and performance overhead to the designers and allow them to select elements which require protection, such as proprietary coefficients. We further showcase the scalability of our complete solution by locking ARM Cortex-M0 microcontroller. We analyze the security promises of the combined RTL solution against oracle-less and oracle-guided attacks with promising results and potential vulnerabilities to explore.

[1]We exclude i2c design from the security analysis due to presence of tri-state gates which are not supported by the open-source attack tools.

References

[1] J. Hurtarte *et al.*, *Understanding Fabless IC Technology*. Elsevier, 2007.
[2] K. Shamsi *et al.*, "IP protection and supply chain security through logic obfuscation: A systematic overview," *ACM TODAES*, 2019.
[3] J. A. Roy *et al.*, "Ending piracy of integrated circuits," *Computer*, vol. 43, no. 10, pp. 30–38, 2010.
[4] B. Shakya *et al.*, "CAS-Lock: A Security-Corruptibility Trade-off Resilient Logic Locking Scheme," *TCHES*, pp. 175–202, 2020.
[5] L. Li and A. Orailoglu, "Piercing logic locking keys through redundancy identification," in *DATE*. IEEE, 2019, pp. 540–545.
[6] A. Sengupta *et al.*, "CAS-Lock: A Security-Corruptibility Trade-off Resilient Logic Locking Scheme," *TCHES*, 2021.
[7] P. Subramanyan *et al.*, "Evaluating the security of logic encryption algorithms," in *HOST*, 2015, pp. 137–143.
[8] A. Rezaei *et al.*, "Rescuing logic encryption in post-sat era by locking & obfuscation," in *DATE*, 2020, pp. 13–18.
[9] N. Limaye *et al.*, "FaSAT- Fault-aided SAT-based attack on Compound Logic Locking Techniques," in *DATE*, 2021, pp. 1–6.
[10] C. Pilato *et al.*, "TAO: Techniques for algorithm-level obfuscation during high-level synthesis," in *DAC*, 2018, pp. 1–6.
[11] C. Pilato *et al.*, "ASSURE: RTL locking against an untrusted foundry," *IEEE Transactions on Very Large Scale Integration Systems*, 2021.
[12] C. Karfa *et al.*, "Is Register Transfer Level Locking Secure?" in *DATE*, 2020, pp. 550–555.
[13] K. Shamsi *et al.*, "KC2: Key-condition crunching for fast sequential circuit deobfuscation," in *DATE*. IEEE, 2019, pp. 534–539.
[14] (2021) FMI+BSO benchmarks. Design For Excellence Lab - NYU AD. [Online]. Available: https://github.com/DfX-NYUAD/FMI_BSO
[15] M. E. Massad *et al.*, "Logic locking for secure outsourced chip fabrication: A new attack and provably secure defense mechanism," *preprint arXiv:1703.10187*, 2017.
[16] P. Chakraborty *et al.*, "SAIL: Machine learning guided structural analysis attack on hardware obfuscation," in *AsianHOST*, 2018, pp. 56–61.
[17] N. Limaye *et al.*, "Thwarting all logic locking attacks: Dishonest oracle with truly random logic locking," *IEEE TCAD*, pp. 1–1, 2020.
[18] R. Karmakar *et al.*, "Encrypt Flip-Flop: A Novel Logic Encryption Technique For Sequential Circuits," *preprint arXiv:1801.04961*, 2018.
[19] X. Wang *et al.*, "Secure Scan and Test Using Obfuscation Throughout Supply Chain," *IEEE TCAD*, vol. 37, no. 9, pp. 1867–1880, 2018.
[20] R. Karmakar *et al.*, "A Scan Obfuscation Guided Design-for-Security Approach For Sequential Circuits," *IEEE TCAS II: Express Briefs*, 2019.
[21] L. Alrahis *et al.*, "ScanSAT: Unlocking Static and Dynamic Scan Obfuscation," *IEEE TETC*, pp. 1–1, 2019.
[22] N. Limaye and O. Sinanoglu, "DynUnlock: Unlocking Scan Chains Obfuscated using Dynamic Keys," in *DATE*, 2020.
[23] (2011) NanGate FreePDK45 Open Cell Library. Nangate Inc. [Online]. Available: http://www.nangate.com/?page_id=2325