The ISC Int'l Journal of Information Security

ISeCure

November 2023, Volume 15, Number 3 (pp. 109–115)

http://www.isecure-journal.org

Presented at the ISCISC'2023 in Tehran, Iran.

HyLock: Hybrid Logic Locking Based on Structural Fuzzing for Resisting Learning-based Attacks **

Mohammad Moradi Shahmiri[1], and Bijan Alizadeh[1,*]

[1]*School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran*

ARTICLE INFO.

Keywords:
Hardware Security, IP Protection, Learning Resilience, Open Source Hardware

Type:
Research Article

doi: 10.22042/isecure.2023.417837.1019

ABSTRACT

The growing popularity of the fabless manufacturing model and the resulting threats have increased the importance of Logic locking as a key-based method for intellectual property (IP) protection. Recently, machine learning (ML)-based attacks have broken most existing locks by exploiting structural traces or undoing optimizations that obfuscate them. A common limitation of these attacks, however, is their reliance on the correlation between the locked circuit structure and the correct key value. In this paper, we introduce structural fuzzing as a simple, nondeterministic, non-optimizing heuristic algorithm that can obfuscate the lock against learning-based attacks, preventing the attacker from predicting the key. We proceed to apply structural fuzzing to multiplexer-based logic locking and propose HyLock, a logic lock with improved resilience against learning-based attacks. In common benchmarks, when compared with a state of the art logic lock, there is on average a 17% decrease in the number of correctly predicted key bits.

1 Introduction

Increases in costs of building and maintaining cutting-edge integrated circuit fabrication facilities have led most companies to go fabless, exposing valuable IP to untrusted parties. Logic locking is a key-based method used for security against an untrusted foundry in which the design is "locked" by adding key gates to the original design. The designer inserts the secret key in a secure on-chip memory; the locked design alone is of no use to the attacker.

In response, two main classes of attacks on logic locking have been proposed: Oracle-less (OL) attacks which assume that the attacker has obtained the reverse engineered locked netlist, and oracle-guided (OG) attacks which assume that the attacker also has access to a copy of the unlocked circuit, for example having procured one from the market. In either case, the designer aims to either recover the correct key and apply it to illegitimate locked copies of the IC, or to remove the lock from the netlist and produce non-locked copies.

As OL attacks make less assumptions, providing a stronger case, we focus the rest of our discussion on them. OL attacks try to either discover the correct key based on structural traces left from the locking

* Corresponding author.

**The ISCISC'2023 program committee effort is highly acknowledged for reviewing this paper.

Email addresses: moradi.shahmiri@ut.ac.ir, b.alizadeh@ut.ac.ir

ISSN: 2008-2045 © 2023 ISC. All rights reserved.

process or to undo that process and expose the circuit to further attacks [1]. Defenses against OL attacks rely on structural changes that conceal the lock. The security of the existing methods for hiding the lock has been questioned, especially in cases that rely on resynthesis as an integral step [2].

In this work, we propose a novel locking method that provides increased resilience against learning-based attacks. We discuss why the existing state of the art defense mechanisms are vulnerable to learning-based attacks on the circuit structure. We then provide structural fuzzing, an algorithm for making nondeterministic, non-optimizing structural modifications while preserving functionality and performance, and demonstrate how these properties lead to increased security. We then use multiplexer-based locking and structural fuzzing to introduce HyLock, a Hybrid Lock that outperforms existing locks in terms of the security and overhead. Hence, our contributions are as follows:

- Providing structural fuzzing as a method for making non-optimizing modifications in circuits and demonstrating how it increases resilience against learning-based attacks while preserving other design parameters.
- Introducing HyLock based on defense mechanisms proposed in the state of the art locks and structural fuzzing.
- Describing an end-to-end design flow that automates the application of HyLock.

The rest of this paper is organized as follows. Section 2 briefly discusses major classes of attacks and defenses against them and provides justification for HyLock. In Section 3 we introduce HyLock and discuss structural fuzzing in detail. In Section 4 we demonstrate the effectiveness of HyLock in terms of security and area overhead. Section 5 discusses the results and concludes the paper.

2 Related Work

In this section we summarize the basics of the cat and mouse game of attacks and countermeasures in the literature, focusing on means and metrics used for provision of security guarantees. We use the observations in this section as the basis for security analysis of structural fuzzing and HyLock. Note that in this work we focus solely on logic locking of combinational circuits.

In its simplest form, logic locking takes place by X(N)ORing a random subset of circuit nets with key inputs. The SAT attack can break this lock by iteratively ruling out wrong keys. Removal attacks try to disable the lock by detecting and fixing the point(s) of the circuit modified during the lock. The classic SAT- and removal- resistant structure provided in the literature is shown in Figure 1, with corruption and restoration modules meant to mitigate for these classes of attacks respectively. Different locks have been proposed based on this principle, including SFLL that corrupts inputs that have a certain Hamming distance from a number of chosen values [3] and CAS lock that utilizes structures based on AND-OR cascades to generate the flip logic [4]. These locks have proven to be effective as defense against most oracle-guided attacks.

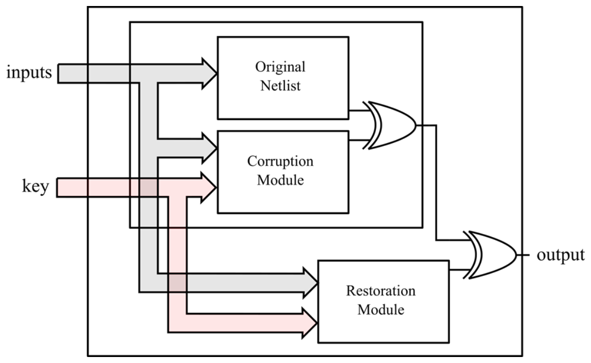

Figure 1. Typical SAT - and removal - resilient lock

The common drawback is that the countermeasures discussed thus far generally rely on synthesis and optimization steps to obfuscate the lock design and protect it. Hence, they are safe so long as the synthesis and optimization steps cannot be reversed by the attacker. But if the attacker can distinguish parts of the circuit as not being modified during the lock process, they can harvest training data from those portions and use them to train a machine learning model that would then proceed to de-obfuscate and expose the lock, as is the case for attacks such as SAIL [5] and OMLA [6]. As most learning-based attacks rely solely on circuit structure and do not need any oracle, they fall into the oracle-less category.

In response to the success of oracle-less attacks on existing logic locks, novel defenses have been proposed. UNSAIL is a learning-based defense against a learning-based attack that gets a locked netlist as input and tries to re-lock it with additional key inputs in a way that the resulting circuit would evade attackers, emulating the attack logic on the defender's side. In practice, the resulting lock acts on subcircuits [7]. Another approach is to lock circuit paths instead of nets or subcircuits, which has been used to great effect in MUX-based locks such as DMUX [2]. Yet another approach is to shift left the locking step in the design process, as used in ASSURE [8], by locking the design at register transfer level (RTL) instead of gate level. Despite the success of these locks in resisting existing attacks, novel attacks such as MuxLink [9] continue to challenge them. Also of concern is the inherent limits to these approaches. Current approaches for enhancing security using machine learning such as UNSAIL tend to target specific approaches towards feature extraction, leading to questionable resilience against other learning-based attacks. Despite the security of RTL locks, the gate level approach is still relevant as it needs to cover a substantially smaller variety of structures and also enables unhampered structural optimization of the design before the locking step. In line with these observations, in the next section we introduce a novel lock that uses multiplexer-based locking and non-deterministic modifications to provide superior security compared to state of the art.

3 Proposed Method

In this paper we propose HyLock, a novel hybrid logic locking method. It gets a gate-level netlist as the input and provides a locked with K key input bits. HyLock is realized in two main steps:

(1) Multiplexer-based locking of circuit paths
(2) Nondeterministic, non-optimizing modifications to conceal the lock gates

In the rest of this section, we describe each HyLock step.

3.1 MUX-Based Locking

HyLock uses a very simple multiplexer-based locking algorithm meant to facilitate its use as a tool for demonstrating the utility of structural fuzzing. Basically, nets that can be intertwined using multiplexers without creating loops in the circuit are randomly chosen and locked, with more weight given to net pairs that would have less adverse effect on circuit timing.

First, static timing analysis (STA) is done for all nets. Then (len(key))/2 pairs of nets are randomly chosen so that the difference between the delays for the nets in each pair is below an acceptable threshold. The nets are also selected so that no two nets in a pair are in the cone of influence of each other. Candidate net pairs for the sample circuit in Figure 2 are (w1, w2), (w1, w3), (w2, w3), (w4, w5).

The chosen net pairs are then randomly scrambled by adding two multiplexers, as shown in Figure 3.

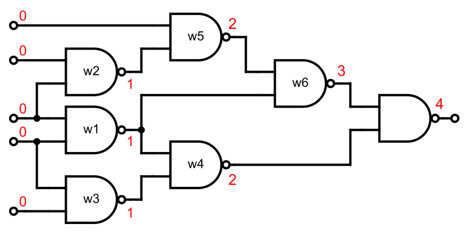

Figure 2. Sample circuit for demonstrating the MUX-based lock step; red digits show net delays

Multiplexers are added in pairs, because the resulting symmetry in the lock structure would decrease the information that can be later inferred by the attacker, increasing lock security [2].

The simplicity of the multiplexer-based locking approach used in HyLock helps us compare the effect of randomness introduced into circuit structure through structural fuzzing versus sophisticated locking algorithms such as DMUX [2]. In some sense, this algorithm is the simplest implementation of a lock that utilizes structural fuzzing.

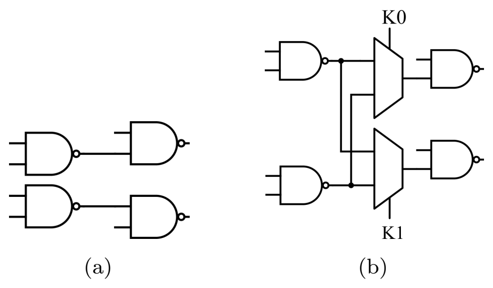

Figure 3. (a) A pair of candidate nets before scrambling using multiplexers, (b) The same pair with lock multiplexers added; K0 and K1 denote the input nets

Algorithm 1 Structural fuzzing.

```
function FUZZ($netlist$, $candidates$, $B_{max}$)
    $breadth \leftarrow 0$
    while size($candidates$) ¿ 0 do
        $gate \leftarrow candidates$.pop()
        $breadth \leftarrow breadth - 1$
        type($gate$) = dual(type($gate$))
        if $gate$ is X(N)OR then
            $nets \leftarrow${a random $gate$ input}
        else
            $nets \leftarrow$set of all $gate$ inputs
        end if
        for $net \in nets$ do
            $netlist$.add($new\_net$ = NOT($net$))
            if $net$ is non-MUX gate output then
                if $breadth < B_{max}$ then
                    $candidates$.push(driver($net$))
                    $breadth \leftarrow breadth + 1$
                end if
            end if
        end for
    end while
    //Cleanup
    for $gate, gate' \in netlist$ do
        Merge $gate$ and $gate'$ if equivalent
        if $gate$ is BUF or NOT then
            Merge $gate$ in its parent if possible
        end if
    end for
    return $netlist$
end function
```

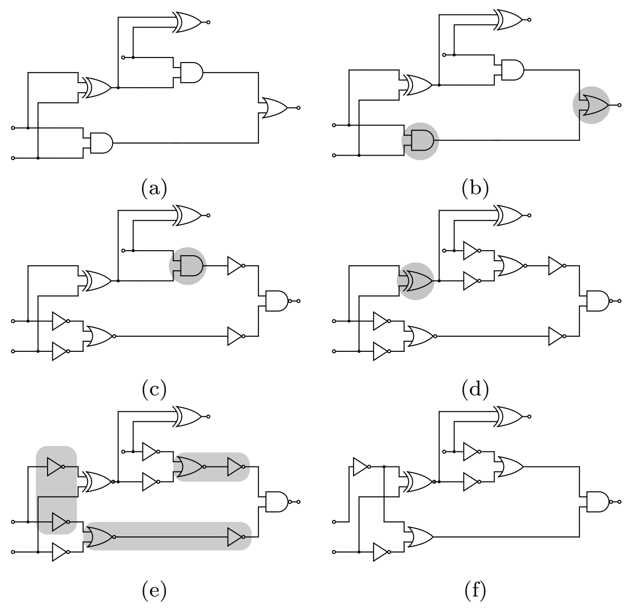

Figure 4. A sample of structural fuzzing, (a) & (b) Candidate gates are shown in red, (c) The cleanup step, (d) Fuzzed circuit

3.2 Structural Fuzzing

The structural fuzzing procedure is described in Algorithm 1. It comprises a modification phase (lines 2-21) and a cleanup phase (lines 23-28). Breadth (line 2) is defined as the number of simultaneous new candidates in the queue in the current step. By setting the maximum breadth parameter, the user can limit the run time and memory requirement. In each modification step, a gate is popped from the front of the candidates queue (line 4) and replaced by its dual gate based on the DeMorgan's laws or its inverted form if it is an X(N)OR (line 6). The inputs are inverted as well to preserve the function of the circuit, becoming new candidates at the queue (lines 12-20). When the candidates reach the primary inputs, the modification phase terminates. During the cleanup phase, all parallel buffers with the same logic function are merged (line 24), and all buffers that can be merged into their previous non-buffer gates are merged (lines 25-27).

Figure 4 illustrates structural fuzzing through an example. Figure 4a, Figure 4b, Figure 4c and Figure 4d show the modification steps. Figure 4e highlights the buffers merged during cleanup. Figure 4f shows the final result. Comparing Figure 4a and Figure 4f, one can see how small modifications alter the structure beyond what is captured in structural modeling of the circuit. Structural fuzzing harnesses this observation to improve lock security, corrupting the training data used for the attack machine learning model.

All learning-based attacks require training data based on functional, baseline circuits. The trainset is harvested either from parts of the circuit expected to be left unchanged by the lock [5, 6, 9] or from the available benchmarks [10]. By decreasing the correlation between the structure and the function, the training data would become unreliable for predicting the changes in structure, hampering key prediction accuracy in the inference phase.

4 Experimental Results

To illustrate the effectiveness and scalability of HyLock, we used it to lock several circuits from ISCAS 85 [11], ISCAS 89 [12] and ITC 99 [13] with 32, 64, 96 and 128-bit keys using both DMUX and HyLock. A brief explanation of the benchmark circuits used in the experiments is shown in Table 1. We compared its security and area overhead with DMUX [2], a state-of-the-art, learning-resilient locking scheme. We used MuxLink [9] to compare security, which uses graph neural networks to predict the correct key in circuits locked using multiplexers and has shown promising results in breaking MUX-based locks. As the methods being investigated are designed for combinational circuits, we consider all the sequential circuits in a single time frame, therefore removing the flip flops and converting their inputs and outputs to primary outputs and primary inputs of the circuit, respectively. In DMUX experiments, we used the same implementation provided in the MuxLink codebase [9]. In the case of HyLock, we used our locking script to lock the designs, then used a custom feature extraction script to derive the necessary features needed for the MuxLink attack. Finally, we ran the attack on all the test examples. For each key bit, MuxLink either 1) predicts the correct value, 2) predicts the wrong value, or 3) fails to predict any value with acceptable certainty. Here we compare the number of

Table 1. A brief introduction of the circuits used in experiments from ISCAS 85 [11], ISCAS 89 [12] and ITC 99 [13]

Family	Benchmark	Gates	Description
ISCAS85	c1908	161	Error detector and corrector
	c5315	603	ALU and selector
	c7552	735	ALU and control
ISCAS89	s1196	267	Combinational circuit with random flip-flops, redundant flip-flops removed from the scan chain
	s1423	295	Not available
	s1494	317	Controller
	s5378	559	Not available
	s9234	762	Real design, partial scan chain
	s13207	1190	Real design, partial scan chain
	s15850	1550	Real design, partial scan chain
	s38417	4688	Real design, partial scan chain
ITC99	b11	266	Scramble string with variable cipher
	b12	506	1-player game (guess a sequence)
	b14	2264	Viper processor (subset)
	b15	3583	80386 processor (subset)

Table 2. HyLock security compared to DMUX, in terms of MuxLink attack resilience [9]

Benchmark	32-bit key		64-bit key		96-bit key		128-bit key		Average improvement (%)
	DMUX	HyLock	DMUX	HyLock	DMUX	HyLock	DMUX	HyLock	
c1908	29	24	56	40	87	46	116	74	29.04
b11	32	28	59	48	87	70	114	104	13.80
s1196	18	16	45	26	66	60	72	74	10.16
s1423	28	28	54	44	85	74	112	70	14.98
s1494	15	18	41	32	58	52	79	62	6.06
b12	27	30	59	48	84	88	117	120	0.33
s5378	28	16	56	40	84	80	111	84	21.94
c5315	31	30	61	54	94	80	122	96	12.24
c7552	32	28	62	50	96	68	127	106	19.21
s9234	31	20	60	46	91	68	117	98	23.77
s13207	25	20	55	32	89	56	113	74	29.11
s15850	28	20	56	48	89	78	108	100	13.80
b14	32	26	64	50	94	66	126	102	22.14
b15	31	28	64	58	94	78	122	116	10.02
s38417	32	22	63	40	90	58	126	70	36.08
								Average	17.51

Table 3. HyLock area overhead relative to pure DMUX, expressed in number of gates after synthesis [2]. Flipflops are unfold before locking, therefore they are not included in the statistics for the locked circuits. Difference is calculated as the increase in overhead for HyLock Compared to DMUX in percents, on average showing a minor improvement

Benchmark	32-bit key		64-bit key		96-bit key		128-bit key		Average improvement (%)
	DMUX	HyLock	DMUX	HyLock	DMUX	HyLock	DMUX	HyLock	
c1908	291	253	366	324	443	410	536	443	31.99
b11	349	364	401	438	491	473	573	568	-2.73
s1196	333	334	376	405	452	459	501	518	-5.06
s1423	362	345	411	392	466	454	519	494	6.19
s1494	378	372	430	417	499	495	564	567	1.58
b12	579	579	646	677	721	720	790	779	-0.93
s5378	647	635	708	687	799	782	852	819	3.71
c5315	727	720	823	818	920	902	947	1001	-1.00
c7552	831	813	1008	862	1080	955	1180	1033	14.83
s9234	873	829	971	914	1043	993	1099	1109	4.63
s13207	1251	1266	1305	1311	1437	1444	1522	1510	-0.34
s15850	1660	1641	1730	1660	1744	1794	1839	1886	-0.13
b14	2561	2390	2879	2480	3066	2539	3155	2612	18.11
b15	3658	3685	3789	3786	3866	3939	4012	4067	-1.07
s38417	4715	4791	4823	4855	4890	4876	4892	5012	-1.14
								Average	5.02

correct values for benchmarks locked using HyLock and the same circuits locked using DMUX. In our experiments, the number of uncertain key bits was generally small (less than 2% of key length) and did not affect the results, so we only report the correct bits here. The results are shown in Table 2. Black and grey bars represent circuits locked with DMUX and HyLock, respectively. As different key lengths were investigated for each circuit, the percentage of correctly predicted key bits has been used as a measure of security. It can be observed that on average, the number of correctly predicted keys has been reduced by above 17%, equivalent to 22 less correct bits for a 128-bit key.

We investigated some examples more thoroughly to understand how the circuit structure predicts the effectiveness of structural fuzzing in protecting it against learning-based attacks. For each circuit net, we defined "depth" as the shortest path between a primary input and that net. We conjectured that higher depths correlate with more modifications in the path and therefore, higher security. We evaluated this conjecture by calculating the average depth for output nets. For s1494, s5378 and s13207, the result is 4.08, 7.00 and 9.39, respectively, supporting the hypothesis. As shown in Figure 5, when average depth is relatively high (above 4), it is positively correlated with improvement in security due to structural fuzzing. For lower values, the effect of other parameters becomes so significant that a single parameter is not sufficient for predicting the outcome. Here, improvement in security is the decrease, in percentage points, of the

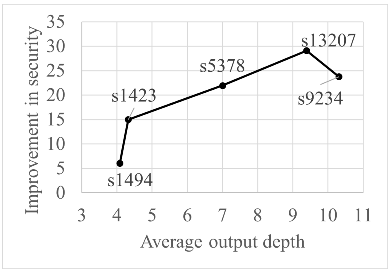

Figure 5. When average output depth is higher than about 4, it is correlated with improvement in security

number of key bits correctly predicted by MuxLink attack on the circuit, when locking it using HyLock instead of DMUX. Another observation is that structural fuzzing tends to be more effective in larger circuits, as they contain longer MUX-free paths that can be modified effectively, also resulting in higher average output depths.

Another vital characteristic of a logic lock is the area overhead introduced to the design. Table 3 lists the area in HyLock-protected designs compared to the same designs locked using DMUX. The numbers listed in the table show the number of gates needed to synthesize the design for the Skywater 130nm HD process node [14] using Yosys [15]. The "Difference" column shows the difference in number of gates between HyLock and DMUX locked gates, with negative numbers pointing to a decrease in the used resources and positive numbers showing an increase. Overall, from the very small improvement in overhead, it can be inferred that HyLock has almost the same impact on circuit area as DMUX.

In closer inspection, the changes in attack accuracy and area overhead can be explained by the random nature of structural fuzzing. Making random changes to circuit structure increases the difficulty of finding the "correct" choice of multiplexer select bits. At the same time, the exact order and nature of optimizations done during synthesis changes at random, which may result in an increase or a decrease in the total number of gates after synthesis. As shown in Table 3, there is on average a slight decrease in overhead, which is due to the cleanup phase of structural fuzzing (Algorithm 1, lines 23 – 28), as it generally results in a decrease in the number of circuit gates.

5 Conclusion

In this work we proposed HyLock, a hybrid lock derived from a combination of multiplexer-based and SAT-resilient locking and non-deterministic heuristics. We provided a simple automated design process for HyLock. We then demonstrated how, despite its simplicity and acceptable overhead, it measurably increases security against novel machine learning-based attacks. We completely based our work on open-source flows, as HyLock is designed to allow low-cost locking of smaller designs with minimum time investment from the designer. The designer can then readily explore the tradeoff between overhead and security by tuning a few global parameters. We also plan to further optimize the lock algorithm and package it as a plugin for Yosys [15] for ease of use.

References

[1] A. Sengupta, N. Limaye, and O. Sinanoglu. Breaking CAS-Lock and its variants by exploiting structural traces. *IACR Trans. Cryptographic Hardware and Embedded Syst.*, pages 418–440, 2021.

[2] D. Sisejkovic, F. Merchant, L. M. Reimann, and R. Leupers. Deceptive Logic Locking for Hardware Integrity Protection Against Machine Learning Attacks. *IEEE Trans. Comput.-Aided Des. Integr. Circuits and Syst.*, pages 1716–1729, 2022.

[3] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. (JV) Rajendran, and O. Sinanoglu. Provably-Secure Logic Locking: From Theory To Practice. In *Proc. 2017 ACM SIGSAC Conf. Comput. and Commun. Secur.*, pages 1601–1618, 2017.

[4] B. Shakya, X. Xu, M. Tehranipoor, and D. Forte. CAS-Lock: A Security-Corruptibility Trade-off Resilient Logic Locking Scheme. *IACR Trans. Cryptographic Hardware and Embedded Syst.*, pages 175–202, 2020.

[5] P. Chakraborty, J. Cruz, A. Alaql, and S. Bhunia. SAIL: Analyzing Structural Artifacts of Logic Locking Using Machine Learning. *IEEE Trans. Inf. Forensics and Secur.*, pages 3828–3842, 2021.

[6] L. Alrahis, S. Patnaik, M. Shafique, and O. Sinanoglu. OMLA: An Oracle-Less Machine Learning-Based Attack on Logic Locking. *IEEE Trans. Circuits and Syst. II: Express Briefs*, pages 1602–1606, 2022.

[7] L. Alrahis, S. Patnaik, J. Knechtel, H. Saleh, B. Mohammad, M. Al-Qutayri, and O. Sinanoglu. UNSAIL: Thwarting Oracle-Less Machine Learning Attacks on Logic Locking. *IEEE Trans. Inf. Forensics and Secur.*, pages 2508–2523, 2021.

[8] C. Pilato, A. B. Chowdhury, D. Sciuto, S. Garg, and R. Karri. ASSURE: RTL Locking Against an Untrusted Foundry. *IEEE Trans. Very Large Scale Integration (VLSI) Syst.*, pages 1306–1318, 2021.

[9] L. Alrahis, S. Patnaik, M. Shafique, and O. Sinanoglu. MuxLink: Circumventing Learning-Resilient MUX-Locking Using Graph Neural Network-based Link Prediction. In *2022 Des., Automat. & Test in Eur. Conf. & Exhib. (DATE)*, pages 694–699, 2022.

[10] D. Sisejkovic, L. M. Reimann, E. Moussavi, F. Merchant, and R. Leupers. Logic Locking at the Frontiers of Machine Learning: A Survey on Developments and Opportunities. In *IFIP/IEEE Int. Conf. Very Large Scale Integration (VLSI-SoC)*, pages 1–6, 2021.

[11] M.C. Hansen, H. Yalcin, and J.P. Hayes. Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering. *IEEE Des. & Test Comput.*, pages 72–80, 1999.

[12] F. Brglez, D. Bryan, and K. Kozminski. Notes on the ISCAS'89 Benchmark Circuits. https://ddd.fit.cvut.cz/www/prj/Benchmarks/iscas89.pdf, 1989. Retrieved January 7, 2023.

[13] F. Corno, M.S. Reorda, and G. Squillero. RT-level ITC'99 benchmarks and first ATPG results. *IEEE Des. & Test Comput.*, pages 44–53, 2000.

[14] Skywater PDK: Open source process design kit for usage with SkyWater Technology Foundry's 130nm node. https://github.com/google/skywater-pdk. Retrieved January 7, 2023.

[15] C. Wolf. Yosys Open SYnthesis Suite. https://yosyshq.net/yosys. Retrieved January 7, 2023.

Mohammad Moradi Shahmiri received the B.Sc. degree in Electrical Engineering from Iran University of Science and Technology, Tehran, Iran, in 2020, the M.Sc. degree in Digital Electronic Systems from the University of Tehran, Iran, in 2023, and is currently pursuing a Ph.D. at the University of Calgary. As a member of the Design, Verification, and Debugging of Embedded Systems (DVDES) at University of Tehran, Mohammad worked on Hardware Security and Logic locking for Hardware Intellectual Property Protection.

Bijan Alizadeh (SM'13) received his Ph.D. degree in electrical and computer engineering from the University of Tehran, Iran, in 2004. He was with the School of Electrical Engineering, Sharif University of Technology, Iran, as an Assistant Professor from 2005 to 2007 and VDEC, The University of Tokyo, Japan, as a Research Associate from 2007 to 2010. In 2011, he joined the School of Electrical and Computer Engineering at the University of Tehran as an assistant professor, where he has been an associate professor since August 2017. He has authored or co-authored over 130 publications in international scientific journals and conferences. He has been engaged in the research and development of VLSI systems, FPGA-based reconfigurable computing, formal verification and debug, hardware security, and high-level synthesis.