Oracle-based Logic Locking Attacks: Protect the Oracle Not Only the Netlist

Emmanouil Kalligeros
Information & Comm. Systems Eng. Dept., University of the Aegean
Samos, Greece
kalliger@aegean.gr

Nikolaos Karousos
Information & Comm. Systems Eng. Dept., University of the Aegean
Samos, Greece
nkarousos@aegean.gr

Irene G. Karybali
Information & Comm. Systems Eng. Dept., University of the Aegean
Samos, Greece
karybali@aegean.gr

***Abstract*—Logic locking has received a lot of attention in the literature due to its very attractive hardware-security characteristics: it can protect against IP piracy and overproduction throughout the whole IC supply chain. However, a large class of logic-locking attacks, the oracle-based ones, take advantage of a functional copy of the chip, the oracle, to extract the key that protects the chip. So far, the techniques dealing with oracle-based attacks focus on the netlist that the attacker possesses, assuming that the oracle is always available. For this reason, they are usually overcome by new attacks. In this paper, we propose a hardware security scheme that targets the protection of the oracle circuit, by locking the circuit when the, necessary for setting the inputs and observing the outputs, scan in/out process begins. Hence, no correct input/output pairs can be acquired to perform the attacks. The proposed scheme is not based on controlling global signals like *test_enable* or *scan_enable*, whose values can be easily suppressed by the attacker. Security threats are identified, discussed and addressed. The developed scheme is combined with a traditional logic locking technique with high output corruptibility, to achieve increased levels of protection.**

***Index Terms*—hardware security, logic locking, oracle-based attacks, scan chains, LFSRs**

I. Introduction

The growing cost of semiconductor fabrication and the high complexity of modern Integrated Circuits (ICs) has transformed ICs' production model in the last two decades [1]. Instead of full in-house development (from concept to chip), some major steps of the manufacturing process, including fabrication, are, most of times, outsourced. However, the cost and time-to-market reduction benefits of this approach are counterbalanced by security vulnerabilities; Intellectual Property (IP) piracy, overproduction, counterfeiting, reverse engineering and hardware Trojan insertion have emerged as significant threats of the semiconductor industry.

Logic locking is a promising hardware security approach that protects against IP piracy and overproduction. Logic locking methods insert some additional hardware (usually key gates in the combinational logic) and a number of extra key inputs into a design. The added locking hardware renders the design unusable if the correct key is not applied to the key inputs. This secret key is written into a tamper-proof memory by the IC designer, after fabrication. One of the important advantages of logic locking is that it can protect against an attacker anywhere in the IC supply chain (SoC integrator, foundry, test, end user). Although both combinational and sequential logic locking techniques exist, in this paper we focus on the combinational ones that, currently, attract a lot of interest in the literature.

The first combinational logic locking techniques [2], [3] were vulnerable to the hill-climbing [4] and key sensitization [5] attacks. For this reason, test-aware [4] and strong logic locking [5] were proposed. However, everything changed with the advent of the SAT attack [6]. The SAT attack employs Boolean satisfiability (SAT) solvers to compute input patterns, which, along with the correct outputs from a functional copy of the chip (called, hereafter, *oracle* circuit) that the attacker possesses, are used to iteratively rule out sets of incorrect keys, until the correct or an equivalent key is identified. Although, key sensitization and hill-climbing attacks were both oracle-based (i.e., they require the correct circuit responses on specific inputs[1], that are acquired by means of the oracle circuit), the SAT attack was (and still is) so powerful, that easily defeated all combinational logic locking techniques of the time. SAT demonstrated the advantages that an adversary has, when a strong attack algorithm is combined with correct responses from the oracle circuit.

After the SAT attack, a tug of war between countermeasures and new attacks has started. The first techniques against SAT attack were SARLock [7] and Anti-SAT [8]. However, the signal probability skew (SPS) attack [9] was proposed to defeat Anti-SAT, while the Double DIP [10] and the Approximate SAT [11] attacks were proposed against SARLock. Moreover, both SARLock and Anti-SAT are vulnerable to removal [9], bypass [12] and bit-flipping [13] attacks. To resist these new attacks, cyclic logic locking [14] was proposed, but it was subsequently overcome by the CycSAT attack [15]. Concurrently with cyclic logic locking, TTLock [16] was proposed that was later generalized to the stripped functionality logic locking (SFLL) technique [17]. However, recently, functional analysis attacks on logic locking (FALL) [18] and other attacks [19] were proposed that defeat SFLL. It is important to note that all the aforementioned attacks, except for FALL, are oracle-based ones. However, FALL is not a general-purpose attack

[1]Actually, the hill-climbing attack can also use the correct responses of the activated circuit to test patterns, which are provided by the designer for manufacturing testing purposes.

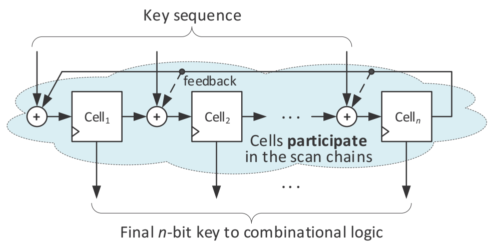

Key sequence
feedback
Cell$_1$
Cell$_2$
Cell$_n$
Cells **participate** in the scan chains
Final *n*-bit key to combinational logic

Fig. 1. Basic version of the proposed oracle protection scheme.

like SAT, but it can be applied only to locking methods that use cube stripping and programmable functionality restoration [18], like [16], [17]. Apart from FALL, some other oracle-less logic locking attacks exist [20]–[22]. In this paper though, we deal with the oracle-based ones, since they constitute a large class of threats against logic locking.

Oracle-based attacks can be completely prevented if the oracle circuit is not available to the attacker. Such attacks on combinational logic locking techniques rely on the circuit's scan operation during testing, so as to gain access to the oracle. Therefore, a simple way to thwart them would be to permanently disable the scan chains (e.g., via fuses), after manufacturing testing and before activation. However, this contradicts the periodic-testing needs of modern ICs [1].

To this end, in this paper we propose a new logic locking scheme that locks the circuit when scan operation begins, thus disabling the attacker from acquiring correct input/output pairs and, hence, performing an oracle-based attack. To the best of our knowledge, no other oracle-protection logic locking scheme has been presented in the literature so far. As this is a new approach on IC protection via logic locking, a possible threat model is identified and various attack scenarios are examined. To respond to these attacks, design guidelines and modifications of the described scheme are proposed. Since the introduced scheme targets oracle protection, it should be combined with one of the existing logic locking schemes to disturb circuit functionality when a wrong key is applied. Therefore, an additional advantage is that a technique that offers high output-corruptibility can be chosen for this purpose. This feature (i.e., output corruption), although important, is not offered by the already existing, state-of-the-art, SAT-resistant logic locking techniques.

II. The Proposed Oracle Protection Scheme

The basic version of the proposed oracle protection (OraP) scheme is shown in Fig. 1. It consists of a key register that is configured as a Linear Feedback Shift Register (LFSR – the reason for this decision will be explained later). Unlocking of a protected chip is a multi-cycle sequential process. During that process, the LFSR is fed with multiple seeds (the term comes from the well-known concept of *reseeding* in testing), which are noted as "Key sequence" in Fig. 1. We call it "key sequence" because it comprises the secret values that the chip owner stores in the tamper-proof memory. However, none of these values is the key that unlocks the circuit. The final n-bit key of the locked combinational circuit is the final state of the register, when the whole key sequence has been fed to it. At that point, the shift operation of the LFSR is disabled and the final key is available at its outputs.

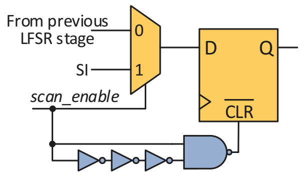


Fig. 2. Circuit of each cell of the key-generating LFSR.

There are a few points that should be clarified about the above described unlocking scheme. First of all, each individual part of the key sequence (i.e., LFSR seed) does not have to be fed into the LFSR right after the other. There can be a number of LFSR free-run cycles between two reseedings (by having stored the all-zero value in a memory position and pushing it to the LFSR in those cycles). Actually, this can also be done after the last seed, as well. Moreover, the number of free-run cycles between two seeds does not have to be constant but can vary. Another issue concerns the volume of the reseeding points (i.e., the XOR gates) inside the LFSR. In Fig. 1, all internal LFSR nodes are used for this purpose, which is the most general case. However, the designer may choose fewer such points. The same holds for the LFSR's feedback: not all internal nodes are connected to it, but only those specified by the characteristic polynomial. This is why the corresponding lines are dashed in Fig. 1.

As mentioned in Sect. I, the main feature of the OraP scheme is that it locks the circuit when testing is applied. To do so, along with every LFSR flip-flop, a pulse generator circuit [23], drawn in blue in Fig. 2, is utilized. This circuit's output is always 1, except for the case that the *scan_enable* signal makes a transition from 0 to 1 (i.e., it is activated). At this point, a 0-pulse is produced by the generator. This pulse resets the flip-flops containing the protected circuit's key. Pulse generators are frequently used in VLSI circuits, even for producing regular clocks in high-end microprocessors [23]. The width of the pulse depends on the length of the generator's inverter chain. In Fig. 2, three inverters are shown, which is a common choice in such circuits. Note that we use a separate pulse generator for every key-register cell and that these cells are connected to the locked circuit's scan chains (as also shown in Fig. 1). These choices have been done for security purposes and will be explained in the following Section. We should also mention that pulse generators are needed only in the key-register cells and not in the rest of the circuit's flip-flops. Finally, since the key register is a special purpose register that is used, apart from testing, only at the beginning of normal circuit operation, no extra reset signal is required. The logic-locking control logic can activate *scan_enable* when normal operation begins (before unlocking), to reset this register.

A. Security Analysis of the OraP Scheme

As mentioned above, all oracle-guided attacks on combinational logic locking techniques, are based on the assumption that access to a functional combinational circuit that is protected by such a technique, is provided through its scan chains. So, scan operation is necessary for these attacks to be applicable. By clearing the key register when scan operation begins, and *before the first scan shift*, we ensure that the oracle circuit will be unavailable when the attacker will try to acquire the correct outputs for specific inputs, via the *scan in – capture – scan out* operation of the circuit.

Mind though that there is a case, in which the oracle circuit scans out a correct response; this response is the last one of the unlocked circuit operation, before the *scan_enable* signal is activated. As a result, one might think that an attacker may be able to determine a primary inputs' sequence that will bring the circuit's state flip-flops to a required (by the employed attack) value during *unlocked* sequential operation, and then activate scan mode to get the corresponding output. However, to do so, the attacker should be able to analyze and simulate the netlist with the correct key values (remember that we refer to an unlocked-mode state sequence). Since the key values are not known, this attack scenario is not possible.

From the above discussion, we deduce that since the OraP locking scheme does not allow the exploitation of the oracle circuit, all oracle-based attacks that require correct circuit outputs are thwarted. Such attacks are the SAT attack and its variants [6], [11], [12] and the hill-climbing attack [4]. Concerning the hill-climbing attack, please recall that instead of a functional circuit, known test responses of the unlocked circuit can be utilized to retrieve the secret key. However, according to the OraP scheme, *the circuit is tested locked* and, as a result, any test responses will correspond to the locked circuit. So, the application of this attack would not reveal the circuit's key. SPS and removal attacks [9] are not applicable to OraP, since the proposed scheme neither has signals with high probability skew, nor by removing the LFSR and/or the key gates that are connected to its outputs, the circuit will unlock. FALL [18] attack is not applicable as well, since OraP does not use cube stripping and programmable functionality restoration. We remind that FALL is an oracle-less attack, targeting specific logic locking techniques. Finally, concerning the key sensitization attack [5], it should be noted that although the LFSR values can be propagated to the circuit's outputs, this can be done in test mode, in which the circuit is locked and the LFSR is reset. Therefore, the valid circuit key will not be revealed.

III. Security Threats and Countermeasures

First of all, let us explain why we avoided a simpler solution that would involve checking and controlling global signals like *test_enable*, *scan_enable* or *reset*. For example, similarly to what we propose, we could check a chip's *test_enable* or *scan_enable* to generate a reset signal to the key register to clear it, when testing or scan operation begins. However, in such a case, an attacker in an untrusted foundry could act as follows: they could fabricate the chip having inserted a Trojan that, when triggered, would suppress the value of the reset signal to the key register. By having done so, they could then buy a functional chip from the open market, trigger the Trojan (e.g., with a specific input sequence) and use scan mode on the unlocked circuit to perform an attack and extract the secret key. It is important to note that the modified by the attacker chip must maintain its original functionality. This is due to the fact that fabricated chips return to their legal owner for activation, which means that they can undergo some standard tests in the owner's trusted environment, while side channel analysis techniques can be also applied to them.

Based on the above threat model, in the following we describe possible attacks to the proposed OraP locking scheme and the corresponding countermeasures. Before proceeding, we should mention that we deal only with attacks at the logic level. To thwart different kinds of attacks, like optical or electrical probing, a multi-layered protection approach should be followed [1]. The OraP scheme serves as one of the layers of such an approach.

a) Suppress the scan_enable signal locally, in every LFSR cell: The attacker may choose to use a Trojan so as to suppress the *scan_enable* signal that resets the LFSR cells. However, since this signal controls the scan operation of those cells as well, the attacker cannot intervene in the stem of *scan_enable* to the LFSR, because it will also disable its scan functionality, as the LFSR is, by design, part of the scan chains. This is actually one of the two reasons for the participation of the LFSR cells in the scan chains. The other one is for improving the fault coverage, as explained in [24] and will be demonstrated in the following Section.

To disable resetting of every LFSR cell, while, at the same time, keeping the circuit's original functionality (see above), the attacker should at minimum replace the NAND2 gate of each pulse generator with a NAND3. The extra input will be driven by the Trojan trigger circuit. Considering an 128-bit key register, which is a common size in logic locking, this modification translates to an overhead of, roughly, 64 NAND2 gates[2]. Mind that this overhead concerns only the Trojan payload, which is on top of the Trojan trigger circuit, and that modern side-channel Trojan detection techniques like [25], can detect very small Trojans in large circuits by using circuit partitioning and transition-fault test patterns. Therefore, the chip owner can apply such a technique to detect the Trojan, after chip activation. To facilitate detection, the LFSR cells could be kept in the same circuit segment, or, at least, should not be evenly distributed in different segments.

In general, with the proposed OraP design and some extra guidelines and modifications that will be discussed shortly, we try to increase the hardware overhead and, hence, the power consumption of a possible Trojan, so as to be detectable by side-channel analysis techniques.

b) Suppress scan_enable for the whole LFSR and exclude it from the scan chains: The attacker, by using a Trojan, may disable the LFSR altogether when in scan mode (so as to keep its state – valid circuit key) and exclude it from the scan chains

[2]Alternatively, the attacker can add a pull-up transistor to the reset input of every LFSR flip-flop, which, due to its increased width, yields similar cost.

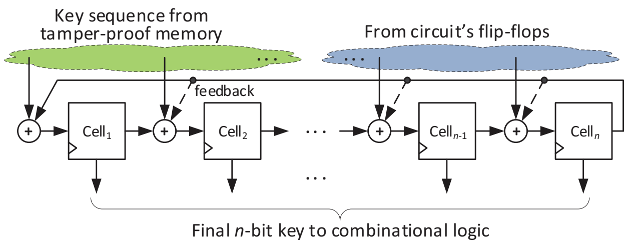
(so as to still be able to apply input vectors and get responses from the circuit). To achieve the latter, a 2-to-1 multiplexer should be placed after every LFSR cell that drives a "normal" circuit flip-flop in the scan chains, so as the LFSR cell to be bypassed. The *select* signal of the multiplexer will be driven by the Trojan trigger circuit.

As a countermeasure that increases the hardware overhead of such a modification, all LFSR cells should be placed, in the design phase, before "normal" circuit flip-flops in the scan chains. In case that multiple LFSR cells have been appointed to the same scan chain, they should be placed in an interleaved manner with "normal" flip-flops. In this way, although the attacker avoids the cost of the modified NAND gate in each pulse generator (since they control, on a single point, the stem of the *scan_enable* signal to the LFSR), they have to insert an additional multiplexer for every LFSR cell. As a result, the final Trojan overhead would be greater than the previous case.

c) Use a shadow register to store the secret key: A shadow register is another option for the attacker. It can be used to store the value of the key register and either apply it to the circuit during testing, or even scan it out through the scan chains. In both cases, n flip-flops are needed for the shadow register, along with n 2-to-1 multiplexers either for connecting the shadow register to the key gates, or for transforming its flip-flops to their scan version, so as to connect them in the scan chains. Therefore, this attack yields a fairly big Trojan payload circuit.

d) Use XOR trees to produce the final key-bit values: It is well-known that XOR gates are linear circuits, that is, at their outputs, linear expressions of their inputs are generated. This feature is extensively exploited in LFSR-based test pattern compression/ decompression. If we assume that the attacker, by analyzing the logic locking control logic, can identify the specific times that LFSR reseedings occur and the number of free-run cycles after feeding every seed to the LFSR, they could replace the unknown key-bit values with binary variables and perform a symbolic simulation of the LFSR (i.e., simulation with variables instead of binary values). At the end of this simulation process, each LFSR cell would contain a linear expression of the variables that have been fed to it (as their name reveals, LFSRs are linear circuits too, since they consist only of memory elements and XOR gates). Consequently, for every LFSR cell, the attacker could construct a XOR tree that would produce the corresponding linear expression, and connect it (through multiplexers) either to the circuit's key gates or to scan cells for direct output.

First of all, let us clarify that this attack requires separate registers for every seed in the key sequence that is fed to the LFSR. This is because only one seed can be at the outputs of the memory every time, but all of them are needed concurrently for feeding the XOR trees. The most important thing though, is that the complexity of the XOR trees depends on the LFSR's characteristic polynomial, the number of seeds fed to the LFSR, the number and positions of reseeding points inside the LFSR, and the number of free-run cycles between reseedings. By choosing these features carefully, the resulting



Key sequence from tamper-proof memory
From circuit's flip-flops
feedback
Cell$_1$
Cell$_2$
Cell$_{n-1}$
Cell$_n$
Final n-bit key to combinational logic

Fig. 3. Modified OraP logic locking scheme.

linear expressions will be complex enough to require big XOR trees for their implementation, which impose significant hardware overhead as a Trojan's payload circuit. This is exactly the reason for utilizing an LFSR as a key register: it can "mix up" the seeds' values and create more complex linear expressions, as compared to a simple shift register.

e) A hardware Trojan to take advantage of the correct scanned-out response: Instead of trying to control the key register, a Trojan may be inserted to control the circuit's "normal" flip-flops, in order to take advantage of the only correct response that the oracle circuit can scan out (refer to Sect. II-A). This can be done in the following way: an input to the oracle circuit is calculated, according to the employed oracle-based attack. This input is shifted into the circuit in test mode. Since in test mode, the LFSR has been already reset. When the shift-in operation is completed, the circuit's mode is changed by the attacker to normal. At this point, the (triggered) Trojan disables the *reset* and *enable* signals of "normal" flip-flops and let the logic locking controller unlock the circuit. After unlocking, the correct key is in the LFSR, while "normal" flip-flops have the desired (by the attack) values, since they have not been reset and updated. The circuit is then let operate for one clock cycle, to capture its response on those values, and then test mode is activated again to scan out the response. Note that in this case, the Trojan payload circuit is fairly small (just a few gates), since a small number of signals need to be checked or controlled to perform the described operation.

To respond to this attack, we propose a modification to the basic OraP scheme of Fig. 1; the basic idea behind this modification is that we *use responses of the locked circuit for the generation of the correct key*. Since, according to the proposed scheme, circuit unlocking is a multi-cycle process, during this process the still locked circuit generates some (wrong) output responses. These responses are normally stored in the circuit's flip-flops. However, the attacker disables these flip-flops for preserving the, necessary for the attack, input value. Nevertheless, if the responses produced during unlocking are fed to the LFSR, they become necessary for unlocking the circuit. As a result, the attacker cannot freeze the flip-flops because, if they do so, wrong values will be fed to the LFSR and the correct key will not be generated.

The modified OraP scheme is shown in Fig. 3. Note that there is no need for extra XOR gates since some of the circuit's flip-flops can be chosen to feed half of the LFSR reseeding points, while the remaining half are driven by the tamper-

TABLE I
HAMMING DISTANCE (HD), AREA AND DELAY OVERHEAD RESULTS

Circuit	# Gates	# Outputs of comb.	LFSR size	# Inputs / Ctrl gate [26]	HD (%)	Ar. Ovhd (%)	Del. Ovhd (%)
s38417	8709	1742	256	3	39.45	33.51	14.29
s38584	11448	1730	186	3	50.00	19.73	0
b17	29267	1512	256	3	35.39	11.21	0
b18	97569	3343	97	5	29.49	1.80	0
b19	196855	6672	208	5	31.00	1.97	4.51
b20	17648	512	236	3	42.27	27.16	21.21
b21	17972	512	229	3	41.00	25.66	19.40
b22	26195	757	243	3	40.37	18.68	18.84

proof memory. This does not compromise the strength of the key sequence, since the same sequence can be applied from half the reseeding points in the double number of cycles. We should also mention that to have better control on the LFSR values, the reseeding points of the key sequence should be interleaved with those from the circuit's flip-flops. This is not shown in Fig. 3 for simplicity. The most interesting feature of the scheme of Fig. 3 is that wrong circuit responses are necessary for unlocking the correct circuit functionality.

IV. EXPERIMENTAL RESULTS

We have combined the proposed OraP scheme with weighted logic locking [26], which is a fault-analysis-based approach with high output corruptibility, and have applied them to the combinational part of the largest ISCAS'89 and ITC'99 benchmark circuits. It is well-known that one of the main disadvantages of SAT-resistant logic locking techniques is that they offer low output corruptibility, a fact that hinders their functional-obfuscation ability [1]. Weighted logic locking combines multiple key inputs in a control gate (AND/NAND) that precedes every XOR/XNOR key gate, increasing this way the actuation probability of the key gates and, hence, output corruptibility. To get output corruption with a high enough Hamming Distance (HD) over correct responses, for circuits with many outputs, in our experiments we set 256 as maximum key size. However, we stopped with smaller key sizes if output corruptibility with HD $= 50\%$ had been achieved, which is the optimal HD value [3], or if output corruptibility, in terms of HD, saturated. Concerning the size of the control gates of weighted logic locking, to reduce hardware overhead we kept them with three inputs, except for the two larger ITC'99 benchmarks, for which we chose control gates with five inputs.

The corresponding results are shown in Table I. In the first three columns, information about the utilized benchmark circuits is provided (name, number of gates without inverters, and number of outputs of their combinational part). The sizes of the employed LFSRs (that equal the corresponding key sizes) are shown in the fourth column, while the number of inputs of the control gates for the application of weighted logic locking, is given in the fifth column. HD, area and delay overhead results are presented in the last three columns of Table I. For calculating HD, the valid and various random keys, along with long pseudorandom input sequences (a few hundreds of thousands of patterns), were applied to the circuits. The ABC synthesis tool [27] was used to estimate the area (in terms of gate count) and delay overhead (in terms of number of levels), after optimizing/resynthesizing both the original and the protected circuit, using the commands *strash* $\rightarrow$ *refactor* $\rightarrow$ *rewrite*, as in [12]. We note that in the results of Table I, we have also taken into account the pulse generator circuits, as well as the XOR gates required for reseeding the LFSRs and for implementing their characteristic polynomials (we used polynomials with a new tap after every eight LFSR cells – this choice leads to high controllability of the LFSR state through reseeding, with relatively low hardware cost). The flip-flops of the LFSRs have not been considered, since the use of key registers is common to all logic locking techniques [1].

As far as HD results are concerned, as can be seen from Table I, high enough values have been achieved (on average, 41.4% for the six smaller circuits and 30.2% for the two larger ones, with too many outputs). Although 50% is the optimal value for HD, as has been shown in [26], for circuits with a great number of outputs, smaller HD values are sufficient to create high ambiguity to the attacker. For example, HD = 31% for b19 means that, on average, 2068 out of its 6672 outputs (not always the same) are corrupted, in the presence of a random key. Output corruption is very important for logic locking, since it renders the design unusable as a black box, which is one of the main objectives of hardware obfuscation [1]. In contrast to state-of-art SAT-resistant logic locking techniques, OraP allows high output corruptibility since it thwarts SAT attack and its variants by disabling the oracle.

Concerning area and delay overheads, although small, they are more noticeable in smaller circuits, due to the fact that we allowed big key sizes, to get output corruptibility as close to the optimal value as possible. The designer, though, can exploit the tradeoff between overhead and output corruptibility for such circuits. However, there is a clear overhead-reduction trend as circuit size increases, while for the largest ITC'99 benchmarks (b18 and b19), which are more representative of large industrial designs, both area and delay overheads are negligible. Please note that 0% delay overhead means that no key gates have been inserted in a circuit's critical path(s).

Since, according to the proposed OraP scheme, the protected circuits are tested locked, we performed a set of experiments to investigate their testability. We employed the Atalanta ATPG tool for generating stuck-at fault test patterns, setting high-effort values to its parameters. Test patterns were generated for both the original version of each circuit and the one protected with OraP + weighted logic locking. The tool was allowed to

TABLE II
STUCK-AT-FAULT COVERAGE (FC) AND REDUNDANT + ABORTED FAULTS RESULTS

Benchmark	Original version		Protected version	
Circuit	*FC (%)*	*# Red.+Abrt faults*	*FC (%)*	*# Red.+Abrt faults*
s38417	99.47	165	99.50	165
s38584	95.85	1506	96.65	1265
b17	97.23	2122	99.08	717
b18	99.43	1513	99.45	1468
b19	99.03	5165	99.21	4254
b20	99.29	324	99.33	318
b21	99.18	381	99.30	340
b22	99.48	352	99.50	346

set any value to the key inputs, since the key register (i.e., the LFSR) is connected to the circuit's scan chains. We note that for b18 and b19, we first used HOPE fault simulator [28] with a large number of pseudorandom patterns to reduce the number of faults handled by Atalanta.

The testability results are shown in Table II. As can be seen, fault coverage improves for the protected version of all examined benchmark circuits. This is explained by the fact that key gates act as control points during testing, while key inputs act as control inputs. So, if key inputs can be set freely when the circuit is in test mode, as in the case of the OraP scheme, then the protected circuit's testability improves. Actually, in Table II we can observe that, although the protected circuits are larger than the original ones, due to the extra control and key gates they contain and, hence, have more faults, the number of faults that are identified as redundant or aborted by the ATPG tool is smaller for the protected circuits. This also demonstrates the improved testability of those circuits.

V. CONCLUSIONS

To remove from the hands of an adversary their most important weapon when deploying an oracle-based attack, the oracle circuit, an oracle protection logic locking scheme, OraP, was presented in this paper. The baseline functionality of the OraP scheme is to self-clear its key register when the circuit's scan operation is enabled, since the adversary gets the required oracle responses by means of this operation. However, to do so in a secure manner, various attack scenarios were identified and analyzed to help shaping the final version of the OraP scheme. An important advantage of OraP is that it can be combined with locking techniques with high output corruptibility, since attacks like SAT and its variants are disabled by the unavailability of the oracle circuit. Therefore, we have ended up with a SAT-resistant scheme that offers high corruption at the circuit's outputs, while, at the same time, it cannot be threatened by any, current or future, oracle-based attack.

REFERENCES

[1] M. T. Rahman *et al.*, "Defense-in-depth: A recipe for logic locking to prevail," *CoRR*, vol. abs/1907.08863, 2019. [Online]. Available: http://arxiv.org/abs/1907.08863v1

[2] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending Piracy of Integrated Circuits," *IEEE Computer*, vol. 43, pp. 30–38, Oct. 2010.

[3] J. Rajendran *et al.*, "Fault analysis-based logic encryption," *IEEE Trans. Comput.*, vol. 64, pp. 410–424, Feb. 2015.

[4] S. M. Plaza and I. L. Markov, "Solving the third-shift problem in IC piracy with test-aware logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, pp. 961–971, June 2015.

[5] M. Yasin, J. Rajendran, O. Sinanoglu, and R. Karri, "On improving the security of logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, pp. 1411–1424, Sept. 2016.

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

[7] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," in *IEEE Int. Symp. on Hardware Oriented Security and Trust (HOST)*, 2016, pp. 236–241.

[8] Y. Xie and A. Srivastava, "Anti-SAT: Mitigating SAT attack on logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, pp. 199–207, Feb. 2019.

[9] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal attacks on logic locking and camouflaging techniques," *IEEE Trans. on Emerging Topics in Computing*, 2017.

[10] Y. Shen and H. Zhou, "Double DIP: Re-evaluating security of logic encryption algorithms," in *Great Lakes Symp. on VLSI*, 2017, pp. 179–184.

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

[12] X. Xu, B. Shakya, M. M. Tehranipoor, and D. Forte, "Novel bypass attack and BDD-based tradeoff analysis against all known logic locking attacks," in *Int. Conf. on Cryptographic Hardware and Embedded Systems*. Springer, 2017, pp. 189–210.

[13] Y. Shen, A. Rezaei, and H. Zhou, "SAT-based bit-flipping attack on logic encryptions," in *Design, Automation & Test in Europe Conference & Exhibition*, 2018, pp. 629–632.

[14] K. Shamsi *et al.*, "Cyclic obfuscation for creating SAT-unresolvable circuits," in *Great Lakes Symp. on VLSI*, 2017, pp. 173–178.

[15] H. Zhou, R. Jiang, and S. Kong, "CycSAT: SAT-based attack on cyclic logic encryptions," in *IEEE/ACM Int. Conf. on Computer-Aided Design*, 2017, pp. 49–56.

[16] M. Yasin *et al.*, "What to lock? functional and parametric locking," in *Great Lakes Symp. on VLSI*, 2017, pp. 351–356.

[17] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: from theory to practice," in *ACM/SIGSAC Conf. on Comp. & Comm. Sec.*, 2017, pp. 1601–1618.

[18] D. Sirone and P. Subramanyan, "Functional analysis attacks on logic locking," in *Design, Automation & Test in Europe Conference & Exhibition*, 2019, pp. 936–939.

[19] F. Yang, M. Tang, and O. Sinanoglu, "Stripped functionality logic locking with hamming distance-based restore unit (sfll-hd) – unlocked," *IEEE Trans. Inf. Forensics Security*, vol. 14, pp. 2778–2786, Oct. 2019.

[20] M. E. Massad, J. Zhang, S. Garg, and M. V. Tripunitara, "Logic locking for secure outsourced chip fabrication: A new attack and provably secure defense mechanism," *CoRR*, vol. abs/1703.10187, 2017. [Online]. Available: http://arxiv.org/abs/1703.10187v1

[21] P. Chakraborty, J. Cruz, and S. Bhunia, "SAIL: Machine learning guided structural analysis attack on hardware obfuscation," in *Asian Hardware Oriented Security and Trust Symposium*, 2018, pp. 56–61.

[22] A. Chakraborty, Y. Xie, and A. Srivastava, "GPU Obfuscation: Attack and defense strategies," in *Design Automation Conference (DAC)*, 2018, pp. 122:1–122:6.

[23] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. Pearson, 4^{th} edition, 2011.

[24] M. Yasin, S. M. Saeed, J. Rajendran, and O. Sinanoglu, "Activation of logic encrypted chips: Pre-test or post-test?" in *Design, Automation & Test in Europe Conference & Exhibition*, 2016, pp. 139–144.

[25] F. S. Hossain, M. Shintani, M. Inoue, and A. Orailoglu, "Variation-aware hardware trojan detection through power side-channel," in *IEEE Int. Test Conference*, 2018, pp. 1–10.

[26] N. Karousos, K. Pexaras, I. G. Karybali, and E. Kalligeros, "Weighted logic locking: A new approach for IC piracy protection," in *IEEE Int. Symp. on On-Line Testing and Robust System Des.*, 2017, pp. 221–226.

[27] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Computer Aided Verification*. Springer Berlin Heidelberg, 2010, pp. 24–40.

[28] H. K. Lee and D. S. Ha, "HOPE: An efficient parallel fault simulator for synchronous sequential circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 15, pp. 1048–1058, Sept. 1996.