Weighted Logic Locking: A New Approach for IC Piracy Protection

Nikolaos Karousos, Konstantinos Pexaras, Irene G. Karybali, and Emmanouil Kalligeros
Information & Communication Systems Engineering Department, University of the Aegean, 83200 Karlovassi, Samos, Greece
nkarousos@aegean.gr, icsd09119@icsd.aegean.gr, karybali@aegean.gr, kalliger@aegean.gr

Abstract—Logic locking has been successfully used for protecting digital circuits against IC piracy. Modern logic locking techniques offer significant security advantages, like high corruptibility of the locked circuit’s outputs when applying random keys (=50% Hamming Distance -HD- compared to the correct outputs), or resilience to the key-sensitization attack. However, there are no techniques to combine both advantages. To solve this problem, weighted logic locking is proposed in this paper. Instead of the conventional single key-input control, the proposed technique uses multiple key-inputs to control every key-gate. This new, weighted key-gate control is by construction immune to the key-sensitization attack, while by employing a new key-gate insertion metric, 50% HD is obtained even for circuits with many outputs. Additionally, weighted key-gate control increases dramatically the probability of any key-gate to corrupt the circuit’s values, which means that fewer key-gates are needed to achieve 50% HD. This way, execution time for locking the circuits is drastically reduced. Apart from these advantages, weighted logic locking is fairly “generic” and can be combined with other techniques to improve security (e.g., to thwart the SAT attack).

I. INTRODUCTION

The shift from in-house fabrication of integrated circuits (ICs) to manufacturing in external foundries worldwide, has led to an explosive increase in the number of counterfeit electronics [1]. Several design houses, even leading-edge ones, outsource IC fabrication and packaging to reduce production cost [2]. This production model, although cost cutting, made hardware vulnerable to various threats [2], [3]: recycled or remarked chips may be sold illegally, IC descriptions for a fab may be stolen, ICs may be reverse engineered, untrusted fabs may overproduce ICs to sell them in the black market, and rogue elements in a foundry may insert hardware Trojan into a design. Since the annual losses of the semiconductor industry, due to counterfeit ICs, are estimated to be very high [3], hardware security has received a lot of attention recently.

Hardware security techniques can be generally classified to passive and active [2], [3]. Passive techniques target piracy detection but they offer no protection against the threats previously mentioned. In the past, only passive hardware security techniques were available [2]. However, various active methods have been recently developed. Logic locking (“logic obfuscation” and “logic encryption” are other terms that have been used in the literature) is a class of active hardware security techniques that offer protection against reverse engineering and overproduction. Logic locking methods insert some additional hardware 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.

Sequential and combinational logic locking methods exist. However, the ability of sequential ones to produce wrong outputs, in the presence of wrong keys, has not been demonstrated [3]. So, more emphasis has been laid on the combinational ones, in which the original circuit is augmented with key-gates. A key-input is also added for every key-gate and is connected to it. Various combinational logic locking techniques have been proposed in the literature, which utilize XOR/XNOR key-gates [2]-[5], AND/OR gates [6], multiplexers [5], [7], or combinations of these gates [8]. The key-inputs are driven by a tamper-evident memory [3]. If a wrong key is fed to the key-inputs, the key-gates corrupt the circuit’s values at their insertion points and, thus, the circuit functions incorrectly.

While the technique of [2], which is known as EPIC (Ending Piracy of ICs), has provided a complete logic locking framework, it suffers from low Hamming distance (HD) between the correct and corrupted circuit responses. A 50% HD value is desirable so as to maximize the ambiguity of an attacker as to the circuit’s output values, when applying random keys to it [4], [5]. Fault analysis (FA)-based logic locking [4], [5] solves this issue at the expense of high execution time of the key-gate insertion algorithm. FA-based logic locking though (as well as EPIC), is vulnerable to the key-sensitization attack [3]. To tackle this problem, strong logic locking was proposed in [3], which, however, does not offer high HD rates. Hence, a technique that combines all the aforementioned advantages is necessary. The proposed weighted logic locking replaces the conventional single key-input control of key-gates with control from multiple key-inputs and offers: a) inherent immunity to the key-sensitization attack, b) 50% HD, with the assistance of a new, effective key-gate insertion metric, and c) significantly improved output-corruption efficiency of the key-gates, which translates to much shorter execution times. To the best of our knowledge, this is the first technique that offers, at the same time, high HD rates and key-sensitization-attack immunity. Moreover, the proposed method is generic enough to be easily combinable with other techniques. This way, resilience to additional attacks (e.g., the SAT [9] or the EPIC attack [7]) can be achieved (Section VI).

II. PREVIOUS WORK AND MOTIVATION

EPIC [2] inserts XOR/XNOR gates in random noncritical locations in a design. Each one of these gates receives a different key-input (hence the name “key-gates”). If a key-input has the correct value (0 for a XOR key-gate and 1 for a XNOR), the corresponding key-gate behaves as a buffer and does not affect the functionality of the original circuit. Otherwise, the key-gate inverts the circuit’s logic value at its insertion point, thus forcing wrong circuit behavior. An example of the application of EPIC to a circuit is shown in Fig. 1. Three key-gates have been inserted in the circuit (KG1, KG2 and KG3, drawn with dashed lines and colored), along with the corresponding key-inputs (K1, K2, K3), while everything else is part of the original circuit. The correct key is {K1, K2, K3} = {0, 0, 1}.

The main disadvantage of EPIC, as explained and demonstrated in [4], [5] is that key-gate insertion in random circuit
locations cannot guarantee a high enough Hamming distance (HD) between the correct circuit outputs and those obtained by applying a wrong key. The optimal HD value is 50%, since this figure maximizes the ambiguity of an attacker as to the circuit’s output values, when random keys are applied to the circuit [4], [5]. To solve the problem of low HD of EPIC, the authors of [4], [5] proposed a fault analysis (FA)-based approach for inserting the key-gates to a circuit. This approach tries to identify, by fault-analysis means, the “best locations” in a circuit to insert a key-gate, i.e., the nodes that, when inverted, affect the greatest number of outputs.

Unfortunately, both EPIC and FA-based logic locking are vulnerable to the key-sensitization attack [3]. A key sensitization attack tries to find an input pattern that sensitizes a key-input to a primary output, without any interference from other key-inputs. Such input patterns can be generated by employing an ATPG tool with X-handling capability (unknown key-inputs are regarded as X’s). The attacker is assumed to have both the netlist of the locked circuit and a functional IC from the open market, so, by applying such patterns (generated by performing ATPG on the netlist) to the functional IC, the targeted key bits can be resolved. Actually, all logic locking techniques that drive key-inputs directly to key-gates are inherently vulnerable to this attack. This is why the authors of [3] proposed strong logic locking, which inserts the key-gates in such locations that propagation of a key-bit is possible only if specific values are set to other key-inputs. Since the key-inputs of the functional IC are not accessible to the attacker [3], key sensitization is restrained. However, due to its key-gate insertion-process characteristics, strong logic locking fails to offer high HD rates [3].

The above discussion reveals that a locking technique, which combines immunity to the inherent weakness of key sensitization, along with the HD effectiveness of FA-based logic locking is necessary. This is not enough though; as its authors report, FA-based logic locking is computationally expensive and its application to large designs may need high execution times [5]. Hence, the required key-sensitization-attack immunity and high HDs should be also paired with improved key-gate efficiency, in terms of output corruption, so as the desired 50% HD figure is achieved with fewer key-gates. This will reduce execution time as well, since the latter is proportional to the number of the added key-gates, as one key-gate is inserted in the circuit in every algorithm’s iteration.

III. WEIGHTED KEY-GATE CONTROL

There are two ways to increase the impact of key-gates to a circuit’s outputs. The first one is to enhance the actuation sensitivity1 of the key-gates when a wrong key is applied to the circuit, and the other is to select nodes that affect as many different outputs as possible during key-gate insertion (Section IV-A). Weighted logic locking exploits both of these ways, in such a fashion that also guarantees immunity to the key-sensitization attack and high HD rates.

When a key-gate is controlled by a single key-input, as it is actually the case in all logic locking approaches in the literature, a random key can actuate the key-gate with a probability of 0.5 ($P_{wor} = 0.5$), since the probabilities of setting the corresponding key-bit to its correct or wrong value are equal (0.5). This means that, on average, only half of the key-gates affect the circuit, when a random wrong key is applied to it. By increasing the $P_{wor}$ of each key-gate we can get more actuated key-gates when applying wrong keys, which in turn means that fewer of them would be required to achieve the desired HD value.

To increase $P_{wor}$ each key-gate should be controlled by more than one key-input. When all the controlling key-inputs get their correct values the key-gate will stay unactuated, otherwise, when even one of them has a wrong value, the key-gate will be actuated (= decoding of the controlling key-inputs’ correct value). To achieve this, an AND or NAND gate should be used along with every key-gate, as shown in Fig. 2. Note that, similar to [2]-[5], weighted logic locking uses XOR/XNOR gates as key-gates.

Compared to Fig. 1, in Fig. 2 there are two key-gates instead of three, but each one of them is accompanied by a control gate (CG1 and CG2). Note also that, although the key-gates are two, the key-inputs remain three, since each key-gate is controlled by two key-inputs; K1 and K2 control KG1 (via CG1), while K1 and K3 control KG2 (via CG2). The correct key is still $\{K1, K2, K3\} = \{0, 0, 1\}$, but KG1 should receive $\{K1, K2\} = \{0, 0\}$ and KG2 should get $\{K1, K3\} = \{0, 1\}$ in order not to be actuated. Therefore, with this arrangement, probability $P_{wor}$ of each key-gate is increased to 0.75, since three out of four controlling key-input combinations will force the key-gate to invert the corresponding circuit’s value. Due to the fact that control gates and the associated $P_{wor}$ resemble the AND/NAND gates and their weights in weighted random pattern generation, we call the proposed approach weighted logic locking.

An example demonstrating the advantages of weighted logic locking is provided in Figures 3 and 4. In Fig. 3, input vector 101100100 is applied to the circuit of Fig. 1, which is locked using conventional single-key-input-controlled key-gates. A wrong key is also applied to the circuit, namely $\{K1, K2, K3\} = \{1, 0, 1\}$. As can be seen in Fig. 3, this key actuates only KG1 (since only the corresponding key-input has a wrong value), the inverted output of which manages to propagate up to circuit output O1. As a result, we get wrong output 001, instead of the correct 101. The HD between the correct and the wrong output is equal to 1/3 (33%).

Fig. 4 shows the application of the same input vector and key to the circuit of Fig. 2, which is protected with weighted

---

1 With the term “actuated key-gate” we mean a key-gate that corrupts the correct circuit’s value at the point of its insertion (but not necessarily at an output). Key-gates can be actuated only in the presence of a wrong key.

2 Actually, they are slightly more than half, since we should exclude the correct-key case, which is however negligible for large key sizes.
logic locking. We can see that both key-gates are actuated, as a consequence of the wrong value on K1 (K1 is a control signal of both KG1 and KG2). The inverted outputs of the key-gates propagate to circuit outputs O1 and O3, yielding an output vector of 000. So, the HD between this (wrong) output vector and the correct circuit response increases to 66% (2/3).

It is worth noting that this increased HD value is achieved with two key-gates in the circuit instead of three in Fig. 3, which is due to the increased $P_{act}$ of KG1 and KG2.

Finally note that weighted logic locking decouples the key-input volume from the number of key-gates inserted in a design. This is important for not compromising security when reducing the overhead imposed on the locked circuits (see Section V). The designer is free to choose which key-inputs to connect to every control-gate (existing, new or a combination of existing and new). Any choice will not compromise $P_{act}$ and, hence, the effectiveness of weighted logic locking in terms of HD.

### A. Immunity to the key-sensitization attack

It is obvious that weighted logic locking is “by construction” immune to the key-sensitization attack. This is a consequence of the fact that key-inputs are not directly driven to key-gates, but are combined in control gates first (i.e., every key-input interferes directly with at least another one). In Fig. 2, for example, to propagate K1 through CG1, the attacker must control K2 and vice versa, while the same holds for the propagation of K1 or K3 through CG2 (K3 and K1, respectively, must be controlled). In general, to propagate any key-bit through any control gate, the rest of the gate’s key-inputs must be controlled. As already mentioned though, the key-inputs of a functional IC are not accessible by the attacker [3], which means that the key-sensitization attack is not applicable to circuits locked using weighted logic locking. Thus, observe that the proposed weighted key-gate control not only improves significantly the actuation probability and, hence, the output-corruption efficiency of the key-gates, but also thwarts the key-sensitization attack.

![Key-gate actuation with conventional logic locking.](image1)

**Fig. 3.** Key-gate actuation with conventional logic locking.

![Key-gate actuation with weighted logic locking.](image2)

**Fig. 4.** Key-gate actuation with weighted logic locking.

### IV. CIRCUIT LOCKING ALGORITHM

The proposed circuit locking algorithm is split into two phases: 1) identification of the best locations (circuit nodes) to insert key-gates, and 2) key- and control-gate insertion until 50% HD is achieved. Phase 1 is shown in Algorithm 1.

As can be seen, in every iteration, the best circuit node to insert a new key-gate is identified by computing a metric’s value (lines 2-5 – the metric itself will be discussed in a while), the circuit’s netlist is updated to include the new key-gate (line 6), and the insertion position is stored in a file (line 7). Execution of Phase 1 concludes when $MaxKeyGates$ circuit nodes have been selected. Unlike what happens in other techniques, Phase 1 spots the best locations to insert the key-gates but does not generate the locked circuit. Key-gate insertion in line 6 is only done for metric-calculation purposes, the updated netlist is discarded at the end of Phase 1, and Phase 1 returns only a log file with the best circuit nodes to insert a key-gate. Key- and control gates are added in Phase 2 that will be explained later. The reason for this split is that we wanted to try out various control-gate sizes (i.e., with different number of inputs), without repeating Phase 1, which is time-consuming. Since, different HD values are obtained with different control-gate sizes and, hence, different volumes of key- and control gates are needed to achieve 50% HD, we separated the processes of location identification (Phase 1) and gate insertion (Phase 2). We note that it is possible to use only some of the locations of Phase 1 in Phase 2, if they suffice to reach the 50% HD target.

#### A. The Proposed Flunexp Metric

The efficiency of Phase 1 depends mainly on the metric used for determining the best circuit nodes for key-gate insertion. We decided to use as basis for the proposed metric, the fault impact metric of [4], [5], since it is very effective in locating the node that, when inverted, affects most of the circuit outputs. To compute fault impact, a set of random patterns should be fault simulated for both single stuck-at-0 (s-a-0) and stuck-at-1 (s-a-1) faults, at all circuit inputs and gate outputs. For each one of these circuit nodes, the volume of patterns that detect the corresponding s-a-0 fault ($NoP_f$), as well as the total number of outputs that get affected by that fault ($NoO_f$) are calculated. Similarly, $NoP_f$ and $NoO_f$ are computed. Then, the fault impact for every circuit node is calculated by the relation: $Fault\, Impact = NoP_f \cdot NoO_f + NoP_f \cdot NoO_f$ [4], [5].

Metrics like fault impact are usually applied iteratively to a partially locked circuit, to get the next best node for key-gate insertion. This is also the case in [4], [5]. However, the iterative application of fault impact has a major drawback, in terms of HD: there is no provision so as the node-selection process not to be trapped in circuit areas that affect already exploited outputs. Consider, for example, a node that influences twelve
circuit outputs and another one that influences only three. If \( \text{NoP}_1 \) and \( \text{NoP}_2 \) are equal for the two nodes, it is obvious that, according to the fault impact metric, the first one will be selected. Note though that if these twelve outputs are also affected by other, already selected nodes, and if this is not the case with the three outputs that are affected by the second node, then, in terms of HD, it is preferable to choose the latter.

Due to this, we introduce a new metric, in which \( \text{NoP}_0, \text{NoO}_0, \text{NoP}_1, \text{NoO}_1 \) are computed by taking into account only the outputs that have not been affected by any of the already selected circuit nodes. We call such outputs "unexploited" and we denote the corresponding quantities as \( \text{NoP}_0\_\text{unexp}, \text{NoO}_0\_\text{unexp}, \text{NoP}_1\_\text{unexp} \) and \( \text{NoO}_1\_\text{unexp} \). Specifically, to count a pattern in \( \text{NoP}_1\_\text{unexp} \) or \( \text{NoP}_1\_\text{unexp} \) it should detect the corresponding fault at an unexploited circuit output, while to count an output, where a fault is detected, in \( \text{NoO}_0\_\text{unexp} \) and \( \text{NoO}_1\_\text{unexp} \) it should be unexploited. We call the new metric unexploited-output-aware fault impact (\( \text{FI}_{\text{unexp}} \) for brevity), and we compute it as follows:

\[
\text{FI}_{\text{unexp}} = \text{NoP}_0\_\text{unexp} \cdot \text{NoO}_0\_\text{unexp} + \text{NoP}_1\_\text{unexp} \cdot \text{NoO}_1\_\text{unexp}
\]

We note that if, during Phase 1, all outputs become exploited, they are all reset to the unexploited state, and the next node selection takes into account all outputs, as in the start of the Phase’s 1 execution (the outputs affected after this point are again marked as exploited, and so on).

### B. Further Optimizations of Phase 1

To increase the efficiency of the node-selection process and reduce execution time, some extra optimizations were implemented. The first one has to do with the circuit locations, for which the \( \text{FI}_{\text{unexp}} \) metric is calculated. Our experiments with the simple fault impact metric of \([4],[5]\) revealed that by including in the examined nodes the outputs of the previously inserted key-gates, as proposed in \([4],[5]\), long runs of key-gates were created, since these locations were repeatedly selected (i.e., the node-selection process was trapped at the same circuit location). This behavior was also verified by the authors of \([4],[5]\) in \([3]\) (refer to Fig. 10 of \([3]\)). Runs of key-gates offer no advantage in terms of HD, since they affect the same circuit location. The employment of the proposed \( \text{FI}_{\text{unexp}} \) metric improves significantly the described behavior, since, to get unexploited outputs, different circuit nodes should be selected. Nevertheless, to totally avoid runs of key-gates, we excluded the outputs of previously inserted key-gates from the next steps of Phase 1.

Our second optimization targets execution time. For speeding up execution, we decided to fault simulate small volumes of random patterns for the computation of the \( \text{FI}_{\text{unexp}} \) metric. Specifically, just 100 random test patterns are fault simulated in every iteration of Phase 1 \([4],[5]\) proposed using 1000 patterns. The experiments we performed showed no performance degradation, in terms of HD. We should note that the metric’s computation dominates the run-time of the whole circuit-locking process. The reason is that, for the computation of either fault impact or \( \text{FI}_{\text{unexp}} \) fault simulation without fault dropping should be performed for most of the circuit’s stuck-at faults (only those at fanout branches are not considered). Hence, as it will be discussed in Section V, the described random-pattern reduction leads to significant execution-time improvements.

### C. Phase 2: Key- and Control-Gate Insertion

Phase 2 is shown in Algorithm 2 and a part of it is rather straightforward: a key- and control-gate pair is inserted in the next of the best circuit locations returned by Phase 1 (line 6), the HD is checked (lines 7–8) and if it is smaller than 50% another key- and control-gate pair is inserted; otherwise, execution of Phase 2 ends. As can be seen, the size of the control gates is given as an input to Phase 2.

However, key-input addition to the processed circuit needs to be discussed. The characteristics of weighted logic locking allow various approaches to be followed. We chose the simplest one, which is to add a new key-input per inserted key-gate (line 4) and to randomly select all the other controlling key-inputs of the gate from the already existing ones (line 5). For example, if we assume control gates of size 3, control gate 10 is driven by key-input 10 and two others randomly selected from the subset of key-inputs 0 to 9. Initially though, we should pre-define, for each circuit, a small set of key-inputs (6 when control gates with 3 inputs are used, 7 when 4-input control gates are used, etc. – line 1), so as to be able to make random key-input selections for the first inserted key-gates. As long as the key-gates are fewer than the initial key-inputs, no extra key-input addition is performed to the circuit, and only those belonging in the initial set are utilized. When the key-gates become as many as the initial key-inputs (line 3), then for each new key-gate, a new key-input is added, as previously explained. This way, the volumes of key-gates and key-inputs are equal at the end of execution of Phase 2. We note that any other key-input insertion approach would not compromise HD, since the probability \( P_{\text{act}} \) of every key-gate depends solely on the size of its control gate.

## V. EVALUATION AND COMPARISONS

For evaluating the effectiveness of weighted logic locking, we implemented the proposed method in Python programming language and performed simulations on a Core i5 PC, using various ISCAS’85, ISCAS’89 and ITC’99 benchmark circuits (the combinational part of the latter two). We should note that the smaller circuits (mainly ISCAS’85) are primarily for comparisons with the state-of-the-art technique in the literature in terms of HD \([4],[5]\), while the larger ones are for demonstrating the effectiveness of weighted logic locking in circuits with more realistic size. We employed the HOPE fault-simulation tool for computing the \( \text{FI}_{\text{unexp}} \) metric for all nodes of the examined circuits in Phase 1, while in Phase 2, after every key- and control-gate pair insertion, we applied the valid and various random keys to calculate the HD between the corresponding outputs. A commercial synthesis tool was used for obtaining the area, power and delay overhead of the proposed method.

The results of the proposed method, as well as insertion-volume comparisons are presented in Table 1. Specifically, in columns 2 to 4 we provide the number of key- and control-gate

---

### Algorithm 2: Phase 2 – Key- and control-gate insertion

**Input:** Netlist, ControlGateSize

1: Insert ControlGateSize+3 key-inputs to \( \{\text{KeyInputs}\} \);
2: do
3: if \( \{\text{KeyGates}\} == \{\text{KeyInputs}\} \) then
4: \( \text{Add new key-input to } \{\text{KeyInputs}\} \);
5: Select key-inputs to drive next control-gate;
6: Insert key- and control gate in next best circuit location; Update Netlist;
7: Compute Hamming Distance (HD);
8: while \( \text{HD} < 50\% \) && (not all locations to insert key-gates are exhausted);

---

### Table 1: Insertion comparison between methods

<table>
<thead>
<tr>
<th>Circuit</th>
<th>NoP</th>
<th>NoO</th>
<th>HD</th>
<th>NoP</th>
<th>NoO</th>
<th>HD</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISCAS'85</td>
<td>100</td>
<td>300</td>
<td>0.7</td>
<td>120</td>
<td>350</td>
<td>0.6</td>
</tr>
<tr>
<td>ISCAS'89</td>
<td>150</td>
<td>400</td>
<td>0.8</td>
<td>180</td>
<td>450</td>
<td>0.7</td>
</tr>
<tr>
<td>ITC’99</td>
<td>200</td>
<td>500</td>
<td>0.9</td>
<td>240</td>
<td>550</td>
<td>0.8</td>
</tr>
</tbody>
</table>

pairs required to achieve 50% HD for every benchmark circuit, using weighted logic locking. We obtained results for the cases of control gates with 3, 4 and 5 inputs (presented in col. 2, 3 and 4, respectively), so as to have a high enough $P_{in}$ in every case. It can be observed that, as expected, as $P_{in}$ increases, fewer insertions are required for achieving the desired 50% HD value.

In Table I we have also included the results for benchmark circuits reported in [4], [5] (FA-based logic locking – the state-of-art logic locking technique in the literature in terms of HD, as mentioned earlier). The best result for each circuit, between those presented in [4] and [5], is shown in the table (a dash in column 5 means that no result for the corresponding circuit has been provided in [4], [5]). FA-based logic locking also achieves 50% HD for all circuits shown, apart from c5315, for which HD reached 48% [5]. It is clear that, compared to FA-based logic locking, the proposed one requires much fewer insertions to reach the 50% HD mark. In fact, in all cases compared in Table I, except for s838 and s9234, weighted logic locking requires from 22.7% to 87.5% fewer insertions to achieve 50% HD. The average insertion-volume reduction, including the results for s838 and s9234, is equal to 53.6%.

Of course, every insertion in weighted logic locking, except for a XOR/XNOR (key-gate), requires an extra control gate (area, power and delay comparisons will follow). However, insertion-volume comparisons are important, since they demonstrate the advantages of the proposed method in terms of key-gate output-corruption efficiency, which translates directly to execution time gains. The achieved insertion-volume reductions result in similar reductions in execution time (see Sect. II), obtained by running Phase 1 with much smaller MaxKeyGates values (the actual run-time reduction is much greater though, as will be discussed shortly). Therefore, weighted logic locking offers simultaneously what [4], [5] (50% HD) and [3] (resilience to the key-sensitization attack) offer separately, and with much shorter execution times as compared to [4], [5].

In Table II we present the area ("Ar.") , power ("Pow.") and delay ("Del.") overhead percentages (%) of weighted logic locking (columns 2-10). The best combined result for every circuit is highlighted. In general, we can see that overheads are reduced as circuit size increases. This is an expected behavior and it can be attributed to the fact that insertion volume does not scale with the size of the locked circuit. In other words, for large circuits, we can achieve 50% HD with relatively few key- and control-gate pairs, as compared to the circuits’ size. Specifically, for the largest benchmarks (c5315, c7552, s5378, s9234, c13207, s15850, s38584, b14 and b15), the average area, power and delay overheads, for the best results of Table II, are 13.8%, 18.6% and 11.3%, respectively, which are fairly small. We will discuss about their further reduction in a while.

Table II contains also the area, power and delay overheads of FA-based logic locking, as reported in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5]. For some smaller circuits (c499-c3540), no separate power and delay overheads were provided in [4], [5].
**VI. RESILIENCE TO ATTACKS AND SECURITY ENHANCEMENTS**

As a member of the family of logic locking techniques, weighted logic locking is resilient to the attacks reported in [2] and [5], like “observing internal signals in running chips” or “correcting the wrong output bits”. Also, as explained in Section III-A, it is “by construction” immune to the key-sensitization attack. Moreover, since weighted logic locking does not alter any of the characteristics of the key-gates apart from \( P_{\text{act}} \) it is fairly “generic” and can be combined with other techniques, irrespective of the used key-gates, to improve security. For example, it can be seamlessly combined with the technique of [7], which deals with EPIC’s vulnerability of “testing after chip activation”; instead of randomly selected, the key-gate insertion nodes of [7] can be chosen according to the \( F_{i-corr} \) metric, while weighted control can be applied to the MUX key-gates of [7].

A serious threat for all logic locking techniques is the satisfiability-checking-based (SAT) attack [9]. The SAT attack employs state-of-the-art SAT solvers to compute input patterns (called distinguishing input patterns), which, along with the correct outputs from the functional IC that the attacker possesses, are used to iteratively rule out sets of incorrect keys, until the correct key or an equivalent to it, is identified. Countermeasures to the SAT attack have been proposed in [3], [10] and [11]. The approach of [10] is not compatible with weighted logic locking, since it is not designed to corrupt simultaneously multiple outputs of a circuit. On the other hand, both the solutions in [3] and [11] are suitable for logic locking methods that alter many circuit outputs. The former ([3]) requires an one-way random function module (e.g., AES) to modify a part of the key; the latter ([11]) uses a fairly lightweight anti-SAT block, whose inputs receive a portion of the key and the value of some nodes of the original circuit (preferably primary inputs), while its output is connected via a XOR or XNOR gate to another node, deeper in the circuit (preferably to one with high observability). Both the solutions of [3] and [11] are perfectly compatible with weighted logic locking and can be employed to thwart the SAT attack.

**VII. CONCLUSIONS**

Weighted logic locking was presented in this paper. We have introduced a new key-gate control scheme with multiple controlling key-inputs per key-gate (weighted control), and a key-gate insertion metric that suits iterative key-gate insertion processes, which target high HD rates. Weighted logic locking offers simultaneously what [4], [5] and [3] offer separately: 50% HD and resilience to the key-sensitization attack. Also, the improved output-corruption efficiency of the key-gates, due to the weighted control, and some additional optimizations of the insertion algorithm, lead to much shorter execution times than those of [4], [5]. The overhead imposed on the locked circuits is fairly small, while the proposed technique can be easily combined with other methods for resisting attacks like SAT or the EPIC attack.

**REFERENCES**


