# ChaoLock: Yet Another SAT-hard Logic Locking using Chaos Computing

Hadi Mardani Kamali*, Kimia Zamiri Azar*, Houman Homayoun†, Avesta Sasan*
*Department of ECE, George Mason University, E-mail: {hmardani, kzamiria, asasan}@gmu.edu
†Department of ECE, University of California, Davis, E-mail: hhomayoun@ucdavis.edu

***Abstract*—Logic locking has been widely evaluated as a proactive countermeasure against the hardware security threats within the IC supply chain. However, the introduction of the SAT attack, and many of its derivatives, has raised big concern about this form of countermeasure. In this paper, we explore the possibility of exploiting chaos computing as a new means of logic locking. We introduce the concept of *chaotic logic locking*, called *ChaoLock*, in which, by leveraging asymmetric inputs in digital chaotic Boolean gates, we define the concept of programmability (key-configurability) to the sets of underlying initial conditions and system parameters. These initial conditions and system parameters determine the operation (functionality) of each digital chaotic Boolean gate. Also, by proposing *dummy* inputs in chaotic Boolean gates, we show that during reverse-engineering, the *dummy* inputs conceal the main functionality of the chaotic Boolean gates, which make the reverse-engineering almost impossible. By performing a security analysis of ChaoLock, we show that with no restriction on conventional CMOS-based ASIC implementation and with no test/debug compromising, none of the state-of-the-art attacks on logic locking, including the SAT attack, could reformulate chaotic Boolean gates while *dummy* inputs are involved and their parameters are locked. Our analysis and experimental results show that with a low number of chaotic Boolean gates mixed with CMOS digital gates, ChaoLock can guarantee resiliency against the state-of-the-art attacks on logic locking at low overhead.**

***Keywords*-Hardware Security, Reverse Engineering, Logic Locking, Chaos Computing**

## I. Introduction

The ever-increasing globalization of the design and implementation of integrated circuits (IC) has enforced the steps of IC manufacturing to be distributed over different facilities to different parties [1]. This is when one or more intellectual properties (IPs) from third-party IP vendors are used to reduce time-to-market, or a post-layout verified design will be sent to a high-tech foundry to be fabricated, or an offshore test entity will be used for test/debug purposes. Due to the lack of trustworthiness and reliable monitoring on the offshore entities, numerous supply chain security threats have raised, such as IC overproduction, Trojan insertion, reverse engineering, IP theft, and counterfeiting [2]. Over the years, amongst different proposed *Design-for-Trust* (*DfTr*) techniques, logic locking [3], [4], as one of the promising solutions, have achieved significant consideration by researchers/designers to combat these threats. By using *Logic locking*, the designer can add programmability (key-programmable gates) into the design using programming values, referred to as the *key*, which are driven by an on-chip tamper-proof non-volatile memory [5], and the content of the memory (the key inputs to the key gates) would be initiated after fabrication via a trusted party. Hence, when the logic locking is in place, the adversary in the manufacturing supply chain cannot recover the correct functionality of the chip without the correct key.

Shortly after the introduction of the primitive logic locking solutions [3], [4], [6], the *Satisfiability* (*SAT*) attack, was proposed [7], [8]. Assuming that the adversary has access to the scan chain, the SAT attack could be applied to each combinational part of the circuit separately. In the SAT attack, the adversary has obtained a reverse-engineered yet locked netlist. Also, (s)he has access to one unlocked/activated chip. Then, as an *oracle-guided* attack, the SAT attack could iteratively rule out the incorrect keys using input queries found by the SAT solver, called discriminating inputs (DIPs).

Since the main strength of the SAT attack comes from the pruning power of each DIP, as well as the access to the scan chain, researchers have investigated and introduced a few main categories in the literature to show the possible countermeasures against this attack:

(1) Some techniques try to exponentially increase the number of required SAT iterations (the number of required DIPs) [9]–[11]. In such techniques, the SAT attack, similar to a brute force attack, faces an exponential runtime. However, new attacks like signal probability skew (SPS), removal, and functional analysis (FALL) attack [12]–[15] show that these techniques suffer from various structural vulnerabilities that were eventually exploited to break them. Besides, due to the structure of obfuscation parts in these techniques, they suffer from very low output corruption that makes them vulnerable to approximate-based attacks [16], [17].

(2) The main aim of some techniques is to make the locked circuit not translatable to a SAT problem, or to trap the SAT attack in an infinite loop, or deceive the SAT attack towards wrong decisions, such as delay locking [18], or cyclic locking [19]–[22]. In these techniques the non-Boolean behavior of the circuit is targeted for obfuscation. However, some of the existing techniques in this category are already broken [23]–[26].

(3) Some techniques try to significantly increase the run-time (the complexity) of *each iteration* of the SAT attack, such as Cross-lock [27] and Full-lock [28]. In such techniques, the wiring (routing) is targeted to be locked

using crossbars or permutation networks. The main idea of this category is to increase the complexity of the SAT circuit that must be solved by the SAT solver. When the complexity of the SAT circuit is high, the number of recursive calls (decisions) on decision tree of the SAT solver would be maximized. However, these techniques are recently broken using new SAT-driven attacks applicable on routing modules [29], [30] suffer from the incurred area that makes them almost impractical for a wide range of circuits.

(4) Apart from categories 1-3, since the SAT attack requires access to the scan chain, some techniques lock/block the scan to restrict any unauthorized access [31]–[35]. The SAT attack is only applicable to combinational circuits; hence, by blocking/locking the scan chain, the adversary is no longer capable of engaging the SAT attack. However, these techniques are broken using unrolling-based SAT attacks as well as the SAT attack integrated with bounded model checkers (BMC) [37]–[39]. Also, blocking the scan chain enforces the tester to rely on the (primary) output pins for any test/debug purpose, which might reduce the test coverage considerably.

In this paper, we explore the characteristics and principles of chaotic Boolean gates to introduce the concept of *chaotic logic locking*, called *ChaoLock*. Since initial conditions and system parameters in basic chaotic Boolean gates determine the operation (functionality) of each gate, in ChaoLock, we engage these underlying parameters as the source of ambiguity for logic locking. In ChaoLock, these underlying parameters must be initiated as the programming values at a trusted party, thus identifying the exact functionality of chaotic Boolean gates would be impossible during fabrication/test or even after reverse engineering.

In ChaoLock, we also exploit the concept of asymmetry in chaotic Boolean gates. Asymmetric chaotic Boolean gates are capable of distinguishing between the permuted input sets, such as {0,1} and {1,0} and therefore can treat asymmetric logic functions correctly. By using asymmetric chaotic Boolean gates, we introduce undetectable *dummy* inputs in chaotic Boolean gates. In ChaoLock, a regular chaotic Boolean gate with $N$ inputs would be transformed to the same gate with $N+d$ inputs, where $d$ is the number of *dummy* inputs. Using *dummy* inputs in asymmetric chaotic Boolean gates will conceal the main inputs of the chaotic Boolean gate. Hence, the adversary is not able to reveal the main functionality of the chaotic Boolean gates. By performing the SAT attack on ChaoLock, we demonstrate that this attack could not scale for compromising designs even while the equivalent SAT model of ChaoLock has been formulated during the attack. Also, we demonstrate that the incapability of the SAT for scaling this approach allows us to easily engage this new locking technique at low overhead with minimal changes.

The rest of this paper is organized as follows. The structure of chaotic Boolean logic has been reviewed in Section II. By listing the threat model in this work in Section III, Section IV shows how we engage parameters and initial conditions as well as asymmetric chaotic Boolean gates as the means of logic locking. Section V presents the results of engaging ChaoLock and its resiliency to the SAT attack. Finally, we conclude this paper in Section VI.

## II. Background

Chaos-based (chaotic) computing is very well-known for the richness of their dynamic. Unlike a conventional FPGA element, where reconfiguration is carried out by switching between multiple single-purpose (static) gates, the most promising feature of chaotic computing is its ability to reconfigure a single chaotic element as different Boolean logic gates (different Boolean functions) [40], [41]. The dynamic morphing from one functionality to another in chaotic computing has received significant attention in recent years [42]. Numerous recent studies have shown the possibility of building different Boolean logic functions using a single chaotic element [41], [43], [44]. Implementing universal NOR gate using Chua's circuit [43], or building *all* $2^{2^2}$ functions of a 2-input Boolean gate using a single chaotic element, called chaogate [41], are the best examples of the realization of digital computing using Chaotic circuits. Also, recently ChaoLogix, Inc. designed and fabricated a proof of concept chip that demonstrates the feasibility of constructing reconfigurable chaotic logic gates in standard CMOS-based VLSI 0.18 $\mu$m process operating at 30 MHz with a 3.1x3.1 mm die size and a 1.8v digital core. More recent study quantitatively shows that chaotic gates could be used for security purposes, where its dynamic structure can help us to mitigate the power analysis based profiling attacks on instructions executed on the circuit [45].

Apart from the richness of their dynamic, the basic structure of chaotic circuits could also grant promising advantages for security purposes. Hence, in the following, we first present how a chaotic Boolean gate could work to yield all fundamental logic functions by *programming* the initial conditions and the parameters of a chaotic system. Then, by introducing asymmetry in these gates, we describe the structure of the chaotic Boolean gate as a new means of logic locking that is strongly resilient against the SAT attack.

### A. General Concept of Chaotic Boolean Gate

Based on the basic theoretical method, for obtaining all $2^{2^2}$ functions of a 2-input gate using the chaotic Boolean gate, the following steps are required to be involved [41]:

(1) The logical inputs $I_1$ and $I_2$ for a 2-input logic gate are encoded by the initial state $x_0$ as $x_0 \rightarrow x_{gate}+X_1+X_2$. $X_1$ (or $X_2$) is a physical quantity that has value 0 when logic input $I_1$ (or $I_2$) is 0, and has value $\delta$ when logic input $I_1$ (or $I_2$) is 1 ($\delta$ is a positive constant). $x_{gate}$ is also a physical value, which can be varied to yield different logic outputs.

Table I
NECESSARY/SUFFICIENT CONDITIONS FOR IMPLEMENTING 2-INPUT BASIC LOGIC GATES WITH THE SAME CHAOTIC BOOLEAN GATE [41].

| Logic Operation | Input Set ($I_1$, $I_2$) | Output | Necessary and Sufficient Condition |
|---|---|---|---|
| AND | $(0,0)$ | 0 | $f(x_{\text{gate}}) < x^*$ |
|  | $(0,1)$ / $(1,0)$ | 0 | $f(x_{\text{gate}}+\delta) < x^*$ |
|  | $(1,1)$ | 1 | $f(x_{\text{gate}}+2\delta) \geq x^*$ |
| OR | $(0,0)$ | 0 | $f(x_{\text{gate}}) < x^*$ |
|  | $(0,1)$ / $(1,0)$ | 1 | $f(x_{\text{gate}}+\delta) \geq x^*$ |
|  | $(1,1)$ | 1 | $f(x_{\text{gate}}+2\delta) \geq x^*$ |
| XOR | $(0,0)$ | 0 | $f(x_{\text{gate}}) < x^*$ |
|  | $(0,1)$ / $(1,0)$ | 1 | $f(x_{\text{gate}}+\delta) \geq x^*$ |
|  | $(1,1)$ | 0 | $f(x_{\text{gate}}+2\delta) < x^*$ |
| NAND | $(0,0)$ | 1 | $f(x_{\text{gate}}) \geq x^*$ |
|  | $(0,1)$ / $(1,0)$ | 1 | $f(x_{\text{gate}}+\delta) \geq x^*$ |
|  | $(1,1)$ | 0 | $f(x_{\text{gate}}+2\delta) < x^*$ |
| NOR | $(0,0)$ | 1 | $f(x_{\text{gate}}) \geq x^*$ |
|  | $(0,1)$ / $(1,0)$ | 0 | $f(x_{\text{gate}}+\delta) < x^*$ |
|  | $(1,1)$ | 0 | $f(x_{\text{gate}}+2\delta) < x^*$ |
| XNOR | $(0,0)$ | 1 | $f(x_{\text{gate}}) \geq x^*$ |
|  | $(0,1)$ / $(1,0)$ | 0 | $f(x_{\text{gate}}+\delta) < x^*$ |
|  | $(1,1)$ | 1 | $f(x_{\text{gate}}+2\delta) \geq x^*$ |

(2) Over $n$ time steps, the chaotic Boolean gate has dynamical evolution, which updates the state of the gate $x \rightarrow f_n(x_0)$, namely, the $n$th iteration of the initial state. Specifically for $n$ = 1, the updated state is $x = f_1(x_0)$.

(3) The output of the chaotic Boolean gate is 0 if $f(x_0) < x^*$, and is 1 when $f(x_0) \geqslant x^*$.

The chaotic system is strongly nonlinear. Hence, a control mechanism is required, such as a threshold controller ($x^*$), to set the initial state ($x_{gate}$ and $x_0$) accurately. Considering $f(x)$ as the dynamic function of the chaotic system, the value of the threshold ($x^*$) and the initial state ($x_{gate}$ and $x_0$) must satisfy the conditions derived from the desired truth table(s). For instance, for building the basic 2-input gates, i.e. *AND*, *OR*, *XOR*, *NAND*, *NOR*, and *XNOR*, the conditions are listed in Table I. For instance, based on the aforementioned steps, for an *AND* gate, when only one of its inputs is 1, then $x_{gate} + X_1 + X_2$ is equal with $x_{gate} + \delta$, and since the output of *AND* gate must be 0 in this case, $f(x_{gate} + \delta)$ must be less than threshold ($x^*$). However, when both inputs are 1, then $x_{gate} + X_1 + X_2$ is equal with $x_{gate} + 2\delta$, and since the output must be 1 in this case, $f(x_{gate} + 2\delta)$ must be equal or greater than ($x^*$). Similarly, the conditions could be formulated for any function.

After formulating the conditions for desired Boolean function, such as conditions for the basic 2-input gates depicted in Table I, the designer must look for the satisfying threshold ($x^*$) and the initial state ($x_{gate}$). For example, a representative example that satisfies the conditions of Table I has been illustrated in Table II, which shows the accurate satisfying values for $x_{gate}$ and the threshold $x^*$. For this example, the logistic equation that provides the dynamical equation is $f(x) = ax(1 - x)$. For this equation, the nonlinearity parameter $a$ has been set to 4 and the constant $\delta$ is set to $1/4$ common to all the logic gates. Note that it is possible to implement the concept with any typical nonlinear function, and there are proof-of-principle realizations of chaos computing using circuits implementing several different nonlinear maps.

Table II
A SPECIFIC SATISFYING SOLUTION FOR THE CONDITIONS IN TABLE I YIELDING THE BASIC 2-INPUT LOGIC GATES ($\delta = 1/4$) [41].

| Operation | AND | OR | XOR | NAND | NOR | XNOR |
|---|---|---|---|---|---|---|
| $x_{\text{gate}}$ | 0 | $1/8$ | $1/4$ | $3/8$ | $5/8$ | $3/4$ |
| $x^*$ | $3/4$ | $11/16$ | $3/4$ | $11/16$ | $5/8$ | $7/16$ |

Table III
CONDITIONS FOR IMPLEMENTING 2-INPUT BASIC LOGIC GATES WITH THE SAME ASYMMETRIC CHAOTIC BOOLEAN GATE [41].

| $\{I_2, I_1\}$ | *fuction* | AND | OR | XOR | NAND | NOR | XNOR |
|---|---|---|---|---|---|---|---|
| $(0,0)$ | $f(x_{\text{gate}})$ | $< x^*$ | $< x^*$ | $< x^*$ | $\geq x^*$ | $\geq x^*$ | $\geq x^*$ |
| $(0,1)$ | $f(x_{\text{gate}}+\delta)$ | $< x^*$ | $\geq x^*$ | $\geq x^*$ | $\geq x^*$ | $< x^*$ | $< x^*$ |
| $(1,0)$ | $f(x_{\text{gate}}+2\delta)$ | $< x^*$ | $\geq x^*$ | $\geq x^*$ | $\geq x^*$ | $< x^*$ | $< x^*$ |
| $(1,1)$ | $f(x_{\text{gate}}+3\delta)$ | $\geq x^*$ | $\geq x^*$ | $< x^*$ | $< x^*$ | $< x^*$ | $\geq x^*$ |

## B. Asymmetric Inputs in Chaotic Boolean Gates

In chaotic Boolean gates, the inputs are encoded via the initial state (step 1) as $x_0 \rightarrow x_{gate} + X_1 + X_2$. Hence, these gates are symmetric by itself, in which with input set either $\{X_1, X_2\}$ = $\{0, 1\}$ or $\{X_1, X_2\}$ = $\{1, 0\}$, the logic output is identical. In fact, the chaotic Boolean gates cannot distinguish the input set $\{X_1, X_2\}$ = $\{0, 1\}$ from the input set $\{X_1, X_2\}$ = $\{1, 0\}$. To support the asymmetric inputs, the chaotic Boolean gates must follow the following steps:

- $X_1 = 0$ when $I_1 = 0$ and $X_1 = \delta$ when $I_1 = 1$.
- $X_2 = 0$ when $I_2 = 0$ and $X_2 = 2\delta$ when $I_2 = 1$.

In this case, the physical value of each input is unique. Assuming this asymmetry, four distinct situations might happen for a 2-input chaotic Boolean gate:

1) $I_1$=0, $I_2$=0 $\rightarrow$ $x_0$=$x_{gate}$+$(0)\delta$+$(0)2\delta$ $\rightarrow$ $x_0$=$x_{gate}$
2) $I_1$=1, $I_2$=0 $\rightarrow$ $x_0$=$x_{gate}$+$(1)\delta$+$(0)2\delta$ $\rightarrow$ $x_0$=$x_{gate} + \delta$
3) $I_1$=0, $I_2$=1 $\rightarrow$ $x_0$=$x_{gate}$+$(1)\delta$+$(0)2\delta$ $\rightarrow$ $x_0$=$x_{gate}$+$2\delta$
4) $I_1$=1, $I_2$=1 $\rightarrow$ $x_0$=$x_{gate}$+$(1)\delta$+$(1)2\delta$ $\rightarrow$ $x_0$=$x_{gate}$+$3\delta$

Using this encoding for the inputs of any chaotic Boolean gate helps us to distinguish between the asymmetric input sets, such as $\{X_1, X_2\}$ = $\{0, 1\}$ and $\{X_1, X_2\}$ = $\{1, 0\}$. Hence, the general formalism for basic chaotic Boolean gate from Table I could be updated to support this asymmetry. Table III shows new conditions formulated while the physical value of inputs are unique (asymmetric).

## C. Chaotic Boolean Gates in Digital Circuit

To build the chaotic Boolean gates, and particularly to guarantee its compatibility/feasibility to be integrated with conventional digital technologies, such as CMOS, there is a well-known 3-transistor CMOS circuit with adjustable non-linear characteristics, which could be used to map discrete-time chaotic signals. As shown in Fig. 1, the circuit has two sets of inputs named as functional input (*in*) and control input (*ctrl*). For implementing $m$-bit input and 1-bit output Boolean function, a $m$-bit digital to analog converter is required, which incurs negligible area overhead. Based on the initial values of *in* and *ctrl*, the output voltage sequence would be able to generate different $m$-bit logic functions. Based on the controlling values, including $v_c$ and $\phi_{1-3}$, the functionality space of this circuit could be maximized to support more functions.

This 3-transistor CMOS-based circuit allows the designers to build it using standard HDL, such as Verilog-A, and integrate it with conventional standard EDA tools. By using this circuit, the fabrication and testing of such chips will be accomplished with no restriction. Also, by using these circuits, the designer has no deal regarding the combination of analog and digital logic for each chaotic gate. Studies on the applications of chaos computing shows 100% compatibility of this technology with standard EDA tools, such as commercial-grade Cadence design [47], resulting in the wide-spread usage of chaotic circuits in different digital circuits []. Moreover, it is worth mentioning that the CMOS circuits that contain chaotic gates require optimization to avoid the variation problems from chip fabrication technologies [48], where a variation in a process, voltage, or temperature (PVT) may degrade or even eliminate the chaotic behavior. However, in many investigations regarding the impact of PVT variation on the functionality of chaotic gates, many meta-heuristic techniques have been introduced showing how numerical methods could be selected to eliminate this concern [49], [50].

Note that as the essence of chaos computing, the sets of initial conditions and system parameters produce a result corresponding to a desired Boolean function. These parameters could be stored and used as a ready *look-up table* (LUT) for future computations using this system. In this case, other combinations of inputs and parameters might lead to results that do not produce a correct mathematical or logical result. Those initial conditions are discarded. So setting the parameters to be the ones that gave the desired functional temporal patterns constitutes programming of the system to give the right output.

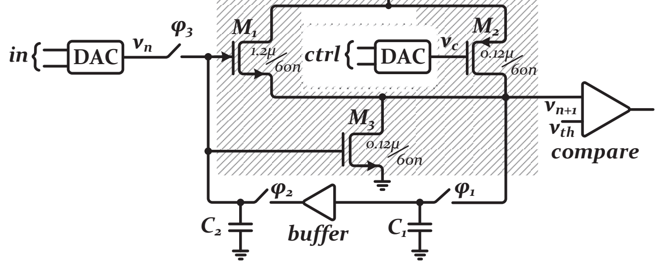


Figure 1. Logic gate built from a discrete time chaotic map circuit (Dudek's Circuit) in a 65 nm CMOS process [46].

## III. Threat Model

We make the following assumption about the adversary capabilities: (1) The adversary has access to a successfully reverse-engineered yet locked netlist. (2) The adversary also has access to an activated/unlocked chip from the market. So (s)he can apply chosen inputs to an unlock IC obtained from the market and collect the correct outputs. (3) The access to the scan chain is *NOT* restricted after activation. The adversary could load any arbitrary value to FFs using SI and read the updated values through SO after the *capture* mode.

## IV. Proposed Scheme: ChaoLock

Our proposed ChaoLock is the realization of the concept of *chaotic logic locking*, which relies on two features of chaotic circuits: (1) the functionality of each chaotic Boolean gate depends on the system parameters and initial conditions that could be stored in look-up-tables (LUTs) as the programming value, and (2) the general formalism of chaotic Boolean gates while their inputs have unique physical value (asymmetric chaotic gates) allows us to have different weighted inputs.

### A. Locked System Parameters/Conditions

As discussed previously, as the essence of chaotic circuits, the sets of initial conditions and system parameters, which determine the functionality, could be stored and used as a ready *look-up table* (LUT). Hence, as a ready to use means of configurability in chaotic circuits, we consider that these parameters, such as $v_c$ and $\phi_{1-3}$ ($x_{gate}$, $x^*$, and $\delta$ in general formalism) would be considered as the post-manufacturing programming values. The exact functionality of each chaotic Boolean gate depends on these parameters. Hence, these system parameters and initial conditions operate as a part of *keys* in ChaoLock.

### B. Dummy Inputs using Asymmetric Chaotic Boolean Gates

By exploiting asymmetric inputs in chaotic Boolean gates, in ChaoLock, a regular chaotic Boolean gate with $N$ inputs is transformed to the same gate with $N + d$ inputs, where $d$ is the number of *dummy* inputs. However, these dummy inputs are not detectable for the adversary.

To add the *dummy* inputs to each chaotic Boolean gate, we need to re-formulate the conditions derived from the desired truth table(s). For this purpose, the chaotic Boolean gates in ChaoLock must follow the following steps, which are almost the same with the formalism of asymmetric inputs with minor changes:

- $X_{d_i} = 0$, when $I_{d_i} = 0$ and $X_{d_i} = \delta$ when $I_{d_i} = 1$.
- $X_1 = 0$ when $I_1 = 0$ and $X_1 = (d + 1)\delta$ when $I_1 = 1$.
- $X_2 = 0$ when $I_2 = 0$ and $X_2 = (d + 2)\delta$ when $I_2 = 1$.

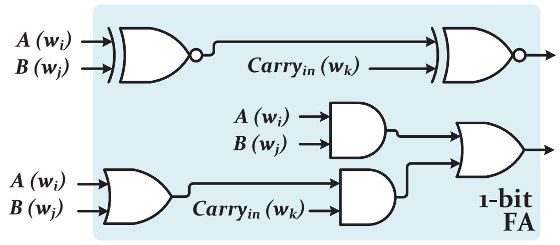
Table IV
CONDITIONS FOR IMPLEMENTING THE 2-INPUT ASYMMETRIC CHAOTIC BOOLEAN GATE WITH $d$ DUMMY INPUTS.

| Functional Inputs | Dummy Inputs | Dynamic *fuction* | AND | OR | XOR | NAND | NOR | XNOR |
|---|---|---|---|---|---|---|---|---|
| $\{I_2, I_1\} = (0, 0)$ | $(0, 0, ..., 0)$ | $f\left(x_{\text{gate}}\right)$ | $<x^*$ (0) | $<x^*$ (0) | $<x^*$ (0) | $\geq x^*$ (1) | $\geq x^*$ (1) | $\geq x^*$ (1) |
| | $(1, 1, ..., 1)$ | $f\left(x_{\text{gate}} + d\delta\right)$ | | | | | | |
| $\{I_2, I_1\} = (0, 1)$ | $(0, 0, ..., 0)$ | $f\left(x_{\text{gate}} + (d+1)\delta\right)$ | $<x^*$ (0) | $\geq x^*$ (1) | $\geq x^*$ (1) | $\geq x^*$ (1) | $<x^*$ (0) | $<x^*$ (0) |
| | $(1, 1, ..., 1)$ | $f\left(x_{\text{gate}} + (2d+1)\delta\right)$ | | | | | | |
| $\{I_2, I_1\} = (1, 0)$ | $(0, 0, ..., 0)$ | $f\left(x_{\text{gate}} + (d+2)\delta\right)$ | $<x^*$ (0) | $\geq x^*$ (1) | $\geq x^*$ (1) | $\geq x^*$ (1) | $<x^*$ (0) | $<x^*$ (0) |
| | $(1, 1, ..., 1)$ | $f\left(x_{\text{gate}} + (2d+2)\delta\right)$ | | | | | | |
| $\{I_2, I_1\} = (1, 1)$ | $(0, 0, ..., 0)$ | $f\left(x_{\text{gate}} + (2d+3)\delta\right)$ | $\geq x^*$ (1) | $\geq x^*$ (1) | $<x^*$ (0) | $<x^*$ (0) | $<x^*$ (0) | $\geq x^*$ (1) |
| | $(1, 1, ..., 1)$ | $f\left(x_{\text{gate}} + (3d+3)\delta\right)$ | | | | | | |

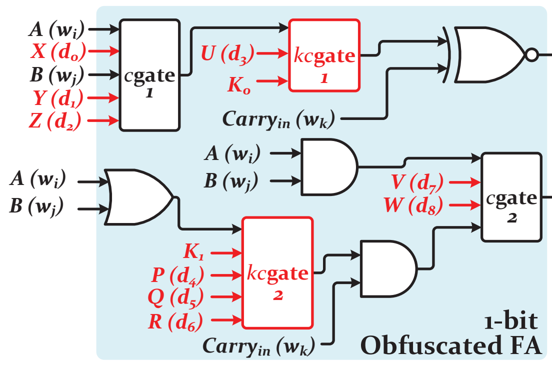
As defined, the physical value of each *dummy* input ($X_{d_i}$) is $\delta$ when the *dummy* input ($I_{d_i}$) is 1. However, the physical value of the functional inputs ($X_{1..m=2}$) is starting from $(d+1)\delta$. By using this model, the physical value of the summation of all *dummy* inputs per each chaotic Boolean gate, which is $\delta$+$\delta$+...+$\delta$=$d\delta$, are less than the physical value of even one of the functional inputs. Hence, if we formulate the threshold controller mechanism as shown in Table IV, none of the *dummy* inputs could change the functionality of the chaotic Boolean gate.

As an example, as shown in Table IV, for a 2-input *OR* gate with $d$ *dummy* inputs, when both functional inputs and all *dummy* inputs are 0, $x_{gate} + X_1 + X_2 + X_{d_1} + ... + X_{d_d}$ is equal with $x_{gate}$, and it must be less than $x^*$. Also, when only both functional inputs are 0, and all *dummy* inputs are 1, then $x_{gate} + X_1 + X_2 + X_{d_1} + ... + X_{d_d}$ is equal with $x_{gate} + d\delta$, and it also must be less than $x^*$. However, when only one of the functional inputs is 1, then $x_{gate} + X_1 + X_2 + X_{d_1} + ... + X_{d_d}$ is equal with $x_{gate} + (d+1)\delta$, and it must be equal or greater than $x^*$ (output must be 1) to operate as an *OR* gate. Similarly, all threshold-based conditions could be formulated for any case of each function.

Since we assumed that the system parameters and the initial conditions in ChaoLock have been stored in LUTs, and these LUTs would be initiated as a post-manufacturing step at a trusted party, the adversary is not able to distinguish between functional inputs and *dummy* inputs. Hence, in ChaoLock, a set of logic gates must be replaced with asymmetric chaotic Boolean gates with *dummy* inputs. As an example, Fig. 2 shows that how a 1-bit full-adder (FA) could be locked using ChaoLock when asymmetric chaotic Boolean gates are in place. In this example, two 2-input gates (one *XNOR* gate and one *OR* gate) are replaced with two multiple-input chaotic Boolean gates ($\{$*XNOR* $\rightarrow$ cgate1$\}$ and $\{$*OR* $\rightarrow$ cgate2$\}$). In this example, cgate1 has 3 *dummy* inputs, and cgate2 has 2 *dummy* inputs. Also, the chaotic Boolean gates could be used as key-programmable gates to add one more level of locking to ChaoLock. For instance, (*k*cgate1 and *k*cgate2) have been inserted in the 1-bit FA with *dummy* inputs in Fig. 2. Hence, retrieving the correct



(a) Full Adder Circuit.



(b) Obfuscated Full Adder using ChaoLock.

Figure 2. ChaoLock on a Full Adder.

functionality of the circuit locked by ChaoLock is not only dependent on the initial conditions and system parameters of chaotic Boolean gates, but it also depends on the value of *key* used as the inputs of *k*cgates.

## C. Resiliency against the SAT attack

ChaoLock could be categorized as an analog-oriented logic locking for digital circuits. However, a big shortcoming of analog-based logic locking techniques for digital circuits is that the adversary can replace any analog-based module (as a black-box) in the locked circuit with a look-up-table (LUT) with the same inputs, and then by engaging the SAT attack, (s)he could find the configuration values of LUTs as the keys. Then, based on the correct key value, the behavior of the analog-based module could be revealed. However, this LUT-based SAT modeling does not work on chaotic Boolean gates when *dummy* inputs are in place.

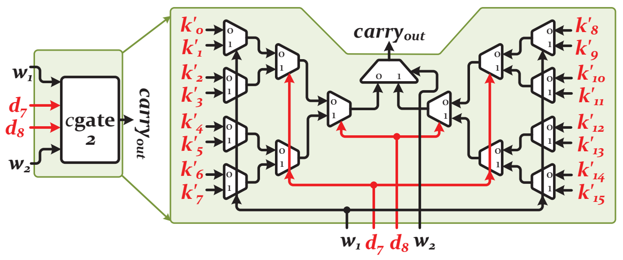

carry_out
w_1
d_7
d_8
w_2
cgate 2
k'_0 k'_1 k'_2 k'_3 k'_4 k'_5 k'_6 k'_7
k'_8 k'_9 k'_10 k'_11 k'_12 k'_13 k'_14 k'_15
w_1 d_7 d_8 w_2

Figure 3. Modeling the chaotic gate using LUT with the Same Inputs.

Assuming that the adversary applies LUT-based SAT modeling on ChaoLock, for 1-bit locked FA in Fig. 2(b) as a case study, (s)he has to replace each chaotic gate with a LUT with same inputs: (1) *c*gate1 in Fig. 2(b) has 5 inputs, and it must be replaced with a LUT with size 5 ($LUT_5$), (2) *c*gate2 has 4 inputs, and it must be replaced with a LUT with size 4 ($LUT_4$), (3) *kc*gate1 has 3 inputs, and it must be replaced with a LUT with size 3 ($LUT_3$), and (4) *kc*gate2 has 5 inputs, and it must be replaced with a LUT with size 5 ($LUT_5$). Each $LUT_n$ could be modeled using a tree of MUXes with depth $n$. As an example, Fig. 3 shows the tree of MUXes (LUT model) for *c*gate2 that has 4 inputs, including 2 dummy inputs.

After replacing each chaotic Boolean gate with its equivalent LUT model with the same inputs, the SAT could be executed to find all keys as well as the configurations of LUTs (e.g. $k'_{0..15}$ for *c*gate2), thus, revealing the correct functionality of the circuit. Then, the adversary can match the correct key and the configurations of LUTs with the chaotic-based locked circuit to find the system parameters and initial conditions. However, in ChaoLock, exploiting *dummy* inputs with negligible area overhead allows us to make it strongly robust to be broken using the SAT attack. In ChaoLock, the size of the equivalent LUT model is equal with the summation of the number of functional inputs + the number of *dummy* inputs. Hence, we could easily engage more *dummy* inputs to increase the size of the equivalent LUT model. When the number of inputs to a chaotic gate is high, the LUT model is alos is a large LUT, and the SAT attack faces exponential execution time increase when LUTs are becoming larger. Fig. 4 shows the execution time of the SAT attack for finding the configuration values of LUTs with different sizes (from 2 to 14). As seen in this *logarithmic* curve, when there is only *ONE* LUT with 14 inputs ($LUT_{14}$), or when there is only *THREE* LUTs with 12 inputs ($LUT_{12}$), or when there is only *FIVE* LUTs with 11 inputs ($LUT_{11}$), the SAT could not find the configuration within $10^6$ Seconds. The exponential regression on these examples shows that the SAT attack could not scale for large LUTs. Hence, ChaoLock could resist against the SAT attack by increasing the number of *dummy* inputs while the area overhead is acceptable. Also, based on our experiments, we demonstrate that although ChaoLock is almost identical to LUT-based logic locking using this modeling, the incurred overhead by ChaoLock is extremely lower.

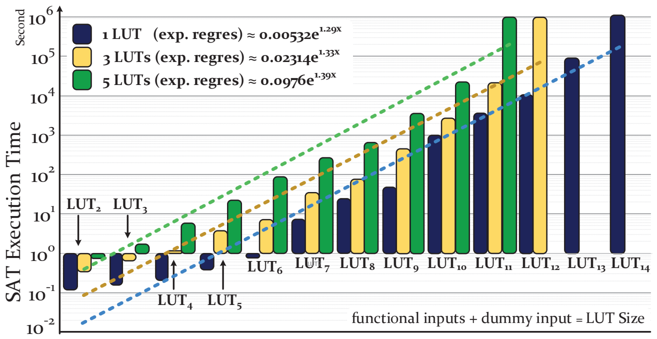


Figure 4. SAT attack exe time on LUT-based Models with Different Sizes.

Table V
SPECIFICATIONS OF THE BENCHMARK CIRCUITS (ISCAS'85, ITC'99, AND WELL-KNOWN ASICS/MICROPROCESSORS).

| Small Circuit: | c432 | c499 | c880 | c1355 | c1908 | c2670 | c3540 | c5315 | c7552 |
|---|---|---|---|---|---|---|---|---|---|
| # of Inputs | 36 | 41 | 60 | 41 | 33 | 233 | 50 | 178 | 207 |
| # of Outputs | 7 | 32 | 26 | 32 | 25 | 140 | 22 | 123 | 108 |
| # of Gates | 160 | 202 | 383 | 546 | 880 | 1269 | 1669 | 2307 | 3513 |

| Large Circuit: | b17 | b18 | b19 | MC8051 | AES-GCM | SPARC |
|---|---|---|---|---|---|---|
| # of Inputs | 37 | 37 | 24 | 52 | 116 | 95 |
| # of Outputs | 97 | 23 | 30 | 112 | 15 | 108 |
| # of Gates | ∼28K | ∼95K | ∼190K | ∼6.6K | ∼49.5K | 233K |

## V. EXPERIMENTAL RESULTS

To evaluate the efficiency expectations of ChaoLock, as discussed previously, we implemented the 3-transistor nonlinear circuit proposed by Dudek *et al.* in a 65nm CMOS process [46]. Feedback circuit, DAC/ADC modules, and the output comparator have been modeled in Verilog-A, and it is integrated with the 3-transistor nonlinear circuit using Cadence Spectre for simulation. The output voltage patterns of the circuit for different control inputs have been evaluated to explore possible logic functions and various options for obtaining the desired function.

To also validate the security expectations, since the chaotic is not translatable directly to a SAT problem, we have developed a straightforward script to remodel the obfuscated circuit based on the LUT-based SAT modeling. Also, to show the resiliency of ChaoLock against the SAT attack, we locked a selected list of combinational ISCAS'85 benchmark circuits, and executed the SAT attack by Subramanyan *et al.* [7] on a 24-core Intel Xeon processors running at 2.4GHz with 256 GB of RAM. Also, all experiments have been done on small ISCAS-85 circuits (For security evaluation), and large ITC'99 circuits, larger ASICs and a well-known microprocessor (For overhead comparison).

Table VI demonstrates the execution time of the SAT attack for different cases on small set of benchmark circuits (ISCAS-85). Since smaller circuits are more vulnerable against such attacks, we used smaller set to depict the

Table VI
THE SAT EXECUTION TIME ON ISCAS-85 CIRCUITS OBFUSCATED USING CHAOLOCK.

| Circuit | Chaotic Contribution = 1% | | | | | | Chaotic Contribution = 2% | | | | | | Chaotic Contribution = 5% | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | {5,6}* | {5,10} | {5,14} | {15,6} | {15,10} | {15,14} | {5,6} | {5,10} | {5,14} | {15,6} | {15,10} | {15,14} | {5,6} | {5,10} | {5,14} | {15,6} | {15,10} | {15,14} |
| c432 | 18.65 | 292.5 | *to*+ | 28.1 | 666.9 | *to* | 159.5 | 1088 | *to* | 541.1 | 2969 | *to* | 1507 | *to* | *to* | 10968 | *to* | *to* |
| c499 | 54.98 | 3138 | *to* | 102.8 | 3098 | *to* | 482.1 | 8992 | *to* | 2706 | 31982 | *to* | 4801 | *to* | *to* | 44291 | *to* | *to* |
| c880 | 84.29 | 21928 | *to* | 294.1 | 8681 | *to* | 1002 | 18924 | *to* | 9521 | *to* | *to* | 10402 | *to* | *to* | *to* | *to* | *to* |
| c1355 | 1024 | *to* | *to* | 27812 | *to* | *to* | 9061 | *to* | *to* | 22911 | *to* | *to* | 29820 | *to* | *to* | *to* | *to* | *to* |
| c1908 | 1258 | *to* | *to* | 36724 | *to* | *to* | 10921 | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* |
| c2670 | 17902 | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* |
| c3540 | 20299 | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* |
| c5315 | 34191 | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* |
| c7552 | 58281 | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* | *to* |

* {#*kc*gates, $d$}: {the number of *key-configurable chaotic* Boolean gates (*kc*gates), the number of *dummy* inputs per each *kc*gate or *c*gate ($d$)}:
+ *to*: timeout, which is set to $10^6$ Seconds, means that the SAT attack could not retrieve the correct functionally within $10^6$ Seconds

strength of ChaoLock in any scenarios. In Table VI, chaotic contribution means that what percentage of logic gates are replaced with chaotic Boolean gates. For instance, 1% chaotic in a *c7552* denotes that 3513/100 $\sim$ 35 gates are replaced with chaotic Boolean gates. In the pair {#*kc*gates, $d$}, (1) the *kc*gates indicates that how many key-programmable chaotic Boolean gate is inserted for locking, and (2) $d$ determines the number of *dummy* inputs for each either *c*gates or *kc*gates. As seen, regardless of the chaotic contribution percentage, for all cases, when 14 *dummy* inputs are added to each chaotic Boolean gate, the SAT could not return the correct key within $10^6$ Seconds. However, when the number of *dummy* inputs is less, the SAT retrieved the correct key while the number of *kc*gates is small. Also, increasing the chaotic contribution enhances the complexity of the SAT model, but the incurred overhead is higher. Hence, increasing the number of *dummy* inputs with negligible area overhead is the best option in ChaoLock.

Table VII shows the area, power, and delay overhead of ChaoLock in two different cases. Both cases are resilient against the SAT attack as shown in Table VI. However, when the chaotic contribution is less, the overhead is considerably lower. Although the overhead is high for moderately small circuits, such as *c432* and *c499*, it could easily be less than 10% for larger circuits such as *c7552*, proving that the incurred overhead is negligible for large-scale ASICs and microprocessors. For this purpose, to show the overhead on larger ASICs and microprocessors, we evaluated ChaoLock on few selected larger circuits. Table VIII reflects the area, delay, and power overhead on these circuits. As shown, for all three cases, area, power, and delay overhead are extremely low. Also, as discussed previously, LUT-based modeling helps the adversary to apply the SAT attack on ChaoLock. It raises a big question that what the advantage of ChaoLock than LUT-based logic locking. Table IX compares the overhead of both techniques (LUT-based and ChaoLock)

Table VII
OVERHEAD PERCENTAGE OF CHAOLOCK (SMALL CIRCUITS).

| Circuit | Original | | | 1% Chaotic + {5,14} | | | 3% Chaotic + {5,10} | | |
|---|---|---|---|---|---|---|---|---|---|
| | a($\mu$m$^2$) | p($\mu$W) | d(ns) | area% | power% | delay% | area% | power% | delay% |
| c432 | 504.5 | 23.9 | 1.3 | 33.4% | 34.9% | 26.5% | 66.2% | 55.8% | 28.4% |
| c499 | 1322.5 | 76.6 | 1.1 | 31.8% | 32.2% | 22.2% | 55.5% | 51.1% | 26.3% |
| c880 | 1049.3 | 57.1 | 1.3 | 26.3% | 27.9% | 19.6% | 49.1% | 47.7% | 22.7% |
| c1355 | 1627.9 | 131.4 | 1.1 | 24.9% | 24.1% | 16.5% | 42.1% | 42.5% | 18.7% |
| c1908 | 1483.7 | 145.8 | 1.5 | 19.1% | 20.8% | 14.1% | 35.7% | 38.7% | 16.5% |
| c2670 | 1704.2 | 138.4 | 1.6 | 16.8% | 17.8% | 12.8% | 26.8% | 30.8% | 15.9% |
| c3540 | 1894.1 | 149.1 | 1.5 | 15.5% | 15.5% | 12% | 21.6% | 26% | 14.5% |
| c5315 | 2168.5 | 167.4 | 1.7 | 12.7% | 11.9% | 10.4% | 18.8% | 17.6% | 12.3% |
| c7552 | 2439.7 | 201.6 | 1.9 | 10.9% | 7.8% | 8.8% | 15.7% | 11.1% | 10.2% |

Table VIII
OVERHEAD PERCENTAGE OF CHAOLOCK (LARGE CIRCUITS).

| Circuit | Original | | | 1% Chaotic + {5,14} | | | 3% Chaotic + {5,10} | | |
|---|---|---|---|---|---|---|---|---|---|
| | a($\mu$m$^2$) | p($\mu$W) | d(ns) | a% | p% | d% | a% | p% | d% |
| b17 | 46872.9 | 1927.6 | 1.34 | 3.1% | 2.4% | 4.1% | 4.5% | 3.7% | 4.9% |
| b18 | 134829 | 2269.9 | 1.82 | 2.7% | 2.1% | 3.5% | 3.9% | 3.3% | 4.3% |
| b19 | 252945 | 2982.7 | 1.97 | 2.4% | 1.9% | 3.1% | 3.5% | 3.1% | 4.0% |
| MC8051 | 4982.9 | 188.6 | 1.29 | 6.8% | 5.5% | 5.9% | 10.2% | 6.8% | 7.1% |
| AES-GCM | 105319 | 1876.4 | 1.76 | 2.9% | 2.2% | 4.0% | 4.1% | 3.7% | 5.4% |
| SPARC | 298231 | 3380.7 | 1.44 | 2.0% | 1.2% | 2.7% | 2.5% | 1.7% | 2.9% |

{a%, p%, d%}: {area%, power%, delay%}

with different numbers of inputs. As shown, The LUT-based[1] Technique increases overhead by more than 10x (at least a factor of 10) even for cases that the LUT size (number of inputs) is less than the number of chaotic gate inputs.

[1]It is implemented based on magnetic tunnel junction (MTJ) using 32nm Synopsys Generic Library [51], [52].

Table IX
OVERHEAD COMPARISON: CHAOLOCK VS. MTJ-LUT LOCKING.

| Circuit | 1% Chaotic + {5,14} | | | $5\times\text{LUT}_{12}$ | | | $5\times\text{LUT}_{14}$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | a% | p% | d% | a% | p% | d% | a% | p% | d% |
| b17 | 3.1% | 2.4% | 4.1% | 98.6% | 101.0% | 37.7% | 3.6x | 3.8x | 52.5% |
| b18 | 2.7% | 2.1% | 3.5% | 85.1% | 92.8% | 31.4% | 3.3x | 3.7x | 48.0% |
| b19 | 2.4% | 1.9% | 3.1% | 77.6% | 84.1% | 26.7% | 3.1x | 3.5x | 42.7% |
| MC8051 | 6.8% | 5.5% | 5.9% | 178.8% | 149.1% | 48.1% | 7.6x | 6.4x | 70.1% |
| AES-GCM | 2.9% | 2.2% | 4.0% | 82.7% | 90.6% | 28.8% | 2.8x | 3.6x | 45.9% |
| SPARC | 2.0% | 1.2% | 2.7% | 65.0% | 72.9% | 22.6% | 2.2x | 2.9x | 36.3% |

$\alpha.\alpha$x: Incurring overhead by $\alpha\alpha$00%
{a%, p%, d%}: {area%, power%, delay%}

## VI. CONCLUSION

In this paper, we introduced a new logic locking scheme using *chaotic circuits*, called ChaoLock. In ChaoLock, we got the benefit of two features of chaotic circuits: (1) The initial conditions and system parameters that determine the functionality of chaotic Boolean gates are used as the source of ambiguity, and (2) By exploiting asymmetric chaotic Boolean gates, we introduced *dummy* inputs in these gates, which conceals the main functionality of each chaotic Boolean gate. We demonstrated that by applying ChaoLock, the locked circuit is resilient against the SAT attack while the incurred area overhead is significantly low.

## ACKNOWLEDGEMENT

This research is supported by the Defense Advanced Research Projects Agency (DARPA-AFRL, #FA8650-18-1-7819), and National Science Foundation (NSF, #1718434).

## REFERENCES

[1] A. Yeh, "Trends in the Global IC Design Market," *DIGITIMES research*, 2012.
[2] M. Rostami *et al.*, "A Primer on Hardware Security: Models, Methods, and Metrics," *Proc. of IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.
[3] J. Roy *et al.*, "Ending Piracy of Integrated Circuits," *Computer*, vol. 43, no. 10, pp. 30–38, 2010.
[4] J. Rajendran *et al.*, "Security Analysis of Logic Obfuscation," in *DAC*, 2012, pp. 83–89.
[5] P. Tuyls *et al.*, "Read-Proof Hardware from Protective Coatings," in *CHES*, 2006, pp. 369–383.
[6] J. Rajendran *et al.*, "Fault Analysis-based Logic Encryption," *IEEE TC*, vol. 64, no. 2, pp. 410–424, 2015.
[7] P. Subramanyan *et al.*, "Evaluating the Security of Logic Encryption Algorithms," in *HOST*, 2015, pp. 137–143.
[8] M. El *et al.*, "Integrated Circuit (IC) Decamouflaging: Reverse Engineering Camouflaged ICs within Minutes," in *NDSS*, 2015, pp. 1–14.
[9] M. Yasin *et al.*, "SARLock: SAT Attack Resistant Logic Locking," in *HOST*, 2016, pp. 236–241.
[10] Y. Xie, and A. Srivastava, "Mitigating SAT Attack on Logic Locking," in *CHES*, 2016, pp. 127–146.
[11] M. Yasin *et al.*, "Provably-Secure Logic Locking: From Theory to Practice," in *ACM CCS*, 2017, pp. 1601–1618.
[12] M. Yasin *et al.*, "Security Analysis of Anti-SAT," in *ASP-DAC*, 2017, pp. 342–347.
[13] M. Yasin *et al.*, "Removal Attacks on Logic Locking and Camouflaging Techniques," *IEEE TETC*, 2017.
[14] X. Xu *et al.*, "Novel Bypass Attack and BDD-based Trade-off against all Known Logic Locking Attacks," in *CHES*, 2017, pp. 189–210.
[15] D. Sirone, and P. Subramanyan, "Functional analysis attacks on logic locking," *IEEE TIFS*, 2020.
[16] K. Shamsi *et al.*, "AppSAT: Approximately Deobfuscating Integrated Circuits," in *HOST*, 2017, pp. 95–100.
[17] Y. Shen and H. Zhou, "Double Dip: Re-evaluating Security of Logic Encryption Algorithms," in *GLSVLSI*, 2017, pp. 179–184.
[18] Y. Xie and A. Srivastava, "Delay Locking: Security Enhancement of Logic Locking against IC Counterfeiting," in *DAC*, 2017, pp. 1–9.
[19] K. Shamsi *et al.*, "Cyclic Obfuscation for Creating SAT-Unresolvable Circuits," in *GLSVLSI*, 2017, pp. 173–178.
[20] A. Rezaei *et al.*, "Cyclic Locking and Memristor-based Obfuscation against CycSAT and Foundry Attacks," in *DATE*, 2018, pp. 85–90.
[21] S. Roshanisefat *et al.*, "SRCLock: SAT-resistant cyclic logic locking for protecting the hardware," in *GLSVLSI*, 2018, pp. 153–158.
[22] A. Rezaei *et al.*, "CycSAT-unresolvable cyclic logic encryption using unreachable states," in *ASP-DAC*, 2019, pp. 358–363.
[23] K. Z. Azar *et al.*, "SMT Attack: Next Generation Attack on Obfuscated Circuits with Capabilities and Performance Beyond the SAT Attacks," *CHES*, pp. 97–122, 2019.
[24] H. Zhou *et al.*, "CycSAT: SAT-based Attack on Cyclic Logic Encryptions," in *ICCAD*, 2017, pp. 49–56.
[25] K. Shamsi *et al.*, "IcySAT: Improved SAT-based Attacks on Cyclic Locked Circuits," in *ICCAD*, 2019, pp. 1–7.
[26] K. Z. Azar *et al.*, "Threats on logic locking: A Decade Later," in *GLSVLSI*, 2019, pp. 471–476.
[27] K. Shamsi *et al.*, "Cross-lock: Dense layout-level interconnect locking using cross-bar architectures," in *GLSVLSI*, 2018, pp. 147–152.
[28] H. M. Kamali *et al.*, "Full-Lock: Hard Distributions of SAT Instances for Obfuscating Circuits using Fully Configurable Logic and Routing Blocks," in *DAC*, 2019.
[29] H. M. Kamali *et al.*, "InterLock: an Intercorrelated Logic and Routing Locking," in *ICCAD*, 2020, pp. 1–9.
[30] K. Z. Azar *et al.*, "NNgSAT: Neural Network guided SAT Attack on Logic Locked Complex Structures," in *ICCAD*, 2020, pp. 1–9.
[31] X. Wang *et al.*, "Secure Scan and Test using Obfuscation throughout Supply Chain," *IEEE TCAD*, vol. 37, no. 9, pp. 1867–1880, 2017.
[32] N. Limaye *et al.*, "Is Robust Design-for-Security Robust Enough? Attack on Locked Circuits with Restricted Scan," in *ICCAD*, 2019, pp. 1–8.
[33] H. M. Kamali *et al.*, "On Designing Secure and Robust Scan Chain for Protecting Obfuscated Logic," *GLSVLSI*, pp. 1–6, 2020.
[34] K. Z. Azar *et al.*, "COMA: Communication and Obfuscation Management Architecture," in *RAID)*, 2019, pp. 181–195.
[35] S. Roshanisefat *et al.*, "DFSSD: Deep Faults and Shallow State Duality, A Provably Strong Obfuscation Solution for Circuits with Restricted Access to Scan Chain," in *VLSI Test Symposium (VTS)*, 2020, pp. 1–6.
[36] H. M. Kamali *et al.*, "SCRAMBLE: The State, Connectivity and Routing Augmentation Model for Building Logic Encryption," *ISVLSI*, pp. 1–7, 2020.
[37] M. El Massad *et al.*, "Reverse Engineering Camouflaged Sequential Circuits without Scan Access," in *ICCAD*, 2017, pp. 33–40.
[38] K. Shamsi *et al.*, "KC2: Key-condition Crunching for Fast Sequential Circuit Deobfuscation," in *DATE*, 2019, pp. 534–539.
[39] L. Alrahis *et al.*, "ScanSAT: Unlocking Obfuscated Scan Chains," in *ASP-DAC*, 2019, pp. 352–357.
[40] S. Sinha and W. L. Ditto, "Dynamics based Computation," *Physical Review Letters*, vol. 81, no. 10, p. 2156, 1998.
[41] W. L. Ditto *et al.*, "Chaogates: Morphing Logic Gates that Exploit Dynamical Patterns," *CHAOS*, vol. 20, no. 3, p. 037107, 2010.
[42] W. L. Ditto *et al.*, "Chaos Computing: Ideas and Implementations," *Philosophical Trans. of the Royal Society A*, vol. 366, no. 1865, pp. 653–664, 2008.
[43] K. Murali *et al.*, "Implementation of NOR Gate by a Chaotic Chua's Circuit," *Int'l Journal of Bifurcation and Chaos*, vol. 13, no. 09, pp. 2669–2672, 2003.
[44] V. Kohar *et al.*, "Implementing Boolean Functions in Digital-Analog Systems," *Physical Review Applied*, vol. 7, no. 4, p. 044006, 2017.
[45] M. B. Majumder *et al.*, "Chaos Computing for Mitigating Side Channel attack," in *HOST*, 2018, pp. 143–146.
[46] P. Dudek and V. D. Juncu, "Compact Discrete-time Chaos Generator Circuit," *Electronics Letters*, vol. 39, no. 20, pp. 1431–1432, 2003.
[47] A. S. Shanta *et al.*, "Design of a reconfigurable chaos gate with enhanced functionality space in 65nm cmos," in *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2018, pp. 1016–1019.
[48] V. H. Carbajal-Gomez *et al.*, "Optimization and CMOS design of chaotic oscillators robust to PVT variations," *Integration*, vol. 65, pp. 32–42, 2019.
[49] J. M. Munoz-Pacheco *et al.*, "Frequency limitations in generating multi-scroll chaotic attractors using CFOAs," *International Journal of Electronics*, vol. 101, no. 11, pp. 1559–1569, 2014.
[50] A. D. Pano-Azucena *et al.*, "FPGA-based implementation of chaotic oscillators by applying the numerical method based on trigonometric polynomials," *AIP Advances*, vol. 8, no. 7, p. 075217, 2018.
[51] H. M. Kamali *et al.*, "LUT-lock: A novel LUT-based logic obfuscation for FPGA-bitstream and ASIC-hardware protection," in *ISVLSI*, 2018, pp. 405–410.
[52] G. Kolhe *et al.*, "Security and Complexity Analysis of LUT-based Obfuscation: From Blueprint to Reality," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.