Generalized SAT-Attack-Resistant Logic Locking

Jingbo Zhou and Xinmiao Zhang, *Senior Member, IEEE*

***Abstract*— Logic locking is used to protect integrated circuits (ICs) from piracy and counterfeiting. An encrypted IC implements the correct function only when the right key is input. Many existing logic-locking methods are subject to the powerful satisfiability (SAT)-based attack. Recently, an Anti-SAT scheme has been developed. By adopting two complementary logic blocks that consist of AND/NAND trees, it makes the number of iterations needed by the SAT attack exponential to the number of input bits. Nevertheless, the Anti-SAT scheme is vulnerable to the later AppSAT and removal attacks. This article proposes a generalized (G-)Anti-SAT scheme. Different from the Anti-SAT scheme, a variety of complementary or non-complementary functions can be adopted for the two blocks in our G-Anti-SAT scheme. The Anti-SAT scheme is just a special case of our proposed design. Our design can achieve higher output corruptibility, which is also tunable, so that better resistance to the AppSAT and removal attacks is achieved. Meanwhile, unlike existing AppSAT-resilient designs, our design does not sacrifice the resistance to the SAT attack.**

***Index Terms*— Anti-SAT, AppSAT attack, hardware security, logic locking, removal attack, SAT attack.**

I. Introduction

NOWADAYS, integrated circuits (ICs) are designed and produced in a multi-vendor environment, which makes the designs face various security threats. In particular, netlists of the ICs may be obtained from reverse engineering or untrusted foundries. IP piracy and counterfeiting cause severe economic loss to the IC designers [1], [2]. IC camouflaging [3], [4] resists reverse engineering [5] by making functionally different logic gates look alike in the layout. However, it does not help in the case that the netlist is known. This article focuses on developing a more secure logic-locking scheme. The basic idea of logic locking is to insert key-controlled logic components into the chip so that the chip does not function correctly without the right key.

Many logic-locking schemes have been developed previously by inserting XOR/XNOR gates [7]–[9], MUX gates [10], [11], or look-up tables (LUTs) [12], [13] controlled by keys. However, these designs can be easily decrypted by the satisfiability (SAT)-based attack [14], which uses Boolean SAT solvers to iteratively update and solve the conjunctive normal form (CNF) formula of the target circuit. In each iteration, a distinguishing input pattern (DIP) is found, and the corresponding correct output is derived by querying the functioning chip. Then the correct output is utilized to exclude wrong keys. For many logic-locking schemes, only a small number of DIPs are needed to exclude all wrong keys. As a result, the SAT attack can be done in short time even if the key size is very large.

Several schemes have been proposed to resist the SAT attack [15], [16]. The main idea is to adopt functional blocks that make the number of iterations and hence the query count in the SAT attack exponential. The Anti-SAT design [15] consists of two complementary function blocks implementing NAND/AND trees. The SAT attack excludes a disjoint set of wrong keys in each iteration and needs to go through all possible input patterns as DIPs before the right key is derived. In SARLock [16], the function blocks are designed so that each DIP can only exclude at most one wrong key. When the number of key bits is larger than the number of input bits, the SAT attack has to enumerate all input patterns. The Anti-SAT and SARLock schemes are vulnerable to more recent attacks. Due to the low corruptibility of all the wrong keys in those schemes, they are subject to the AppSAT attack [17]. Additionally, the AND/NAND functions in the two blocks of the Anti-SAT scheme lead to large signal skew, which makes this scheme also subject to the signal probability skew (SPS) attack [18].

Combining a traditional high output error scheme, such as [7], [8], [10], with a SAT-resilience block, such as the Anti-SAT or SARLock, can increase the overall output corruptibility. However, the compound scheme can be reduced to standalone SAT-resilient block, from which the AppSAT attack [17] can recover an approximate key. Also, the AppSAT-guided removal (AGR) attack [18] and bit-flip attack [19] can separate the key inputs for high error output and the key inputs for the SAT-resistant block. Then using structural analysis, the output signal of the SAT-resilient block can be identified and set to constant to make the circuit function correctly.

It was claimed in [17], [20] that there is a fundamental trade-off between the corruptibility of a logic-locking block and number of queries needed in the SAT attack. Increasing the corruptibility will always reduce the query count. The stripped-functionality logic locking (SFLL) [21] increases the corruptibility by adopting Hamming distance checkers of higher weight. The diversified tree logic (DTL) [17] and its special case, error-controllable encryption (ECE) [22], replace the AND gates in an AND tree structure by XOR/OR/NAND gates to control the true set of the tree output signal. The DTL can be also incorporated in the Anti-SAT [15] and SARLock [16] schemes. All of these schemes increase the corruptibility of

Manuscript received January 4, 2020; revised May 19, 2020, December 11, 2020, and January 21, 2021; accepted February 1, 2021. Date of publication February 12, 2021; date of current version February 26, 2021. This work was supported in part by the U.S. Air Force Research Laboratory under Award FA8650-20-C-1719. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Tim Güneysu. *(Corresponding author: Jingbo Zhou.)*

The authors are with the Department of Electrical & Computer Engineering, The Ohio State University, Columbus, OH 43210 USA (e-mail: zhou.2955@osu.edu; zhang.8952@osu.edu).

Digital Object Identifier 10.1109/TIFS.2021.3059271

each wrong key at the cost of reduced query count. Besides, the SFLL is subject to the functionality analysis attack [23] and the structural analysis attack [24], both of which try to analyze the crucial components in the SFLL to get the right keys.

This article proposes a generalized (G-) Anti-SAT scheme. The proposed design not only resists the SAT attack, but also improves the approximate resiliency without sacrificing the query count. Besides, it has better resistance to removal attacks. Different from the previous schemes, the wrong key set that can be excluded by each DIP in our design has a unique wrong key that is not included in the wrong key sets of any other DIPs. This property makes the design have exponential query count and hence always resistant to the SAT attack. At the same time, the wrong key sets can have overlaps in order to increase the corruptibility. Compared to the Anti-SAT scheme [15], our proposed design is generalized in two dimensions. The two functions do not have to be AND/NAND or complementary. The Anti-SAT design is just a special case of our proposed G-Anti-SAT scheme. The cascaded(CAS)-lock logic-locking block [25] that is implemented with cascaded AND/OR gates can also increase the corruptibility without sacrificing the resistance to the SAT attack. It is also a special case of our proposed design.

The major contributions of this article are as follows.

1) Generalized constraints on the two function blocks are proposed to make the number of SAT attack queries exponential to the input size.
2) Following the constraints, logic-locking blocks can be designed to increase the corruptibility without sacrificing the query count. Design procedures using K-maps are given for the G-Anti-SAT block to achieve higher corruptibility and at the same time exponential query count.
3) The generalized constraints allow a variety of functions to be used in the logic-locking block. The functions do not have to be AND/NAND or complementary. The variations of functions allow better resistance to attacks based on structural or function analysis.
4) The non-AND/NAND function and higher corruptibility of our design allow better resistance to the removal attacks [18], bypass attack [27] and bit-flip attack [19].

This article is organized as follows. Section II briefly introduces available attacks and the Anti-SAT design. Section III proposes our G-Anti-SAT constraints. Section IV presents methodologies for developing functions satisfying the G-Anti-SAT constraints using K-maps. Analysis and experimental results showing the resistance of our design to various attacks are given in Section V. Discussions and conclusions follow in Section VI and Section VII, respectively.

II. Background

This section introduces basic knowledge about the SAT attack, Anti-SAT block, AppSAT and removal attacks.

A. SAT Attack

The SAT attack [14] is a powerful technique against logic-locking. The attack model assumes that the attacker has access to the gate-level netlist of the locked circuit, which can be obtained by reverse engineering or from an un-trusted foundry. Represent the netlist of the locked circuit by $Y = f_e(X, K)$, where X, K, and Y are the primary input, key input, and primary output vectors, respectively. Its CNF formula is represented as $C_e(X, K, Y)$. It is also assumed that the attacker has an activated chip with the right key input, whose function is denoted by $Y = f_o(X)$. Queries can be made on the chip to find the correct outputs for given inputs.

The SAT attack finds the right key by excluding all wrong keys through utilizing DIPs. Initially, a SAT solver is applied to the following formula

$$F_0 := C_e(X, K_1, Y_1) \wedge C_e(X, K_2, Y_2) \wedge (Y_1 \neq Y_2) \quad (1)$$

to solve for an X that leads to different outputs, Y_1 and Y_2, under two different keys, K_1 and K_2. This X is referred to as a DIP and is denoted by X_1^d. Then the activated chip is queried to get the corresponding correct output $Y_1^d = f_o(X_1^d)$. After that, new constraints according to X_1^d and Y_1^d are added and the SAT formula in (1) is updated as $F_1 = F_0 \wedge C_e(X_1^d, K_1, Y_1^d) \wedge C_e(X_1^d, K_2, Y_1^d)$. Then the updated SAT formula is solved for another DIP X_2^d, which is used to query the activated chip and accordingly update the SAT formula. This process is repeated iteratively. In the ith iteration, the SAT formula is

$$F_i = F_0 \bigwedge_{j=1}^{i} (C_e(X_j^d, K_1, Y_j^d) \wedge C_e(X_j^d, K_2, Y_j^d)).$$

If F_i is satisfiable, then there exist at least one pair of keys K_1, K_2, and X_{i+1}^d such that $f_e(X_{i+1}^d, K_1) \neq f_e(X_{i+1}^d, K_2)$, which means not all wrong keys have been excluded from the key space. When the SAT formula is no longer satisfiable in an iteration, say λ, the algorithm stops. At this time, the right key can be derived by solving the following SAT formula

$$F := \bigwedge_{i=1}^{\lambda} C_e(X_i^d, K, Y_i^d).$$

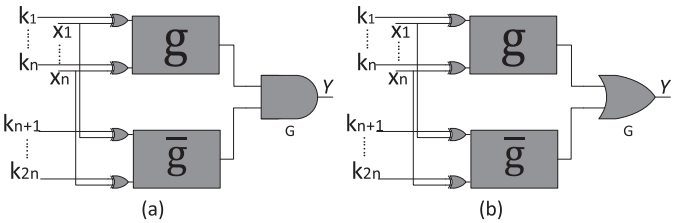

Fig. 1. Anti-SAT block. (a) Type-0 block; (b) Type-1 block.

B. Anti-SAT Block

The Anti-SAT block is proposed in [15]. It is composed of two complementary functions g and $\bar{g}$ as shown in Fig. 1. These two functions share the same input X but have different keys. The outputs of the two functions can be ANDed or ORed to generate the output as shown in Fig. 1 (a) and (b), respectively. They are referred to as the type-0 and type-1 blocks, respectively. Let $K_g = [k_1, k_2, \cdots, k_n]$ and $K_{\bar{g}} = [k_{n+1}, k_{n+2}, \cdots, k_{2n}]$. Any $K_g = K_{\bar{g}}$ are right keys for the Anti-SAT scheme. Since g and $\bar{g}$ are complementary functions,

the correct output of the type-0 block in Fig. 1(a) is '0', and that of the type-1 block in Fig. 1(b) is '1'.

Let $X = [x_1, x_2, \cdots, x_n]$. The input to the g function in Fig. 1(a) is $L = X \oplus K_g$. Define

$$\begin{aligned} \mathbf{G}^T &= \{L|g(L) = 1\}, \quad (|\mathbf{G}^T| = p) \\ \mathbf{G}^F &= \{L|g(L) = 0\}, \quad (|\mathbf{G}^F| = 2^n - p) \end{aligned} \tag{2}$$

In the remainder of this article, $\mathbf{G}^T$ is referred to as the true set. In [15], it has been derived that the total number of iterations needed by the SAT attack on the structure shown in Fig. 1 is lower-bounded by

$$\lambda \geq \frac{2^{2n} - 2^n}{p \times (2^n - p)}.$$

When $p = 1$ or $2^n - 1$, $\lambda \geq 2^n$. Since there are 2^n input patterns, this means that the number of iterations needed by the SAT attack and hence the number of queries to the chip is 2^n and all input patterns need to be gone through as DIPs to find the right keys. In this case, the SAT attack is effectively resisted. A natural candidate for g that satisfies $p = 1$ or $p = 2^n - 1$ is the AND or NAND of all inputs.

C. AppSAT Attack

Logic-locking schemes with low corruptibility can be decrypted by the AppSAT attack [17], which avoids exponential number of iterations by introducing random query reinforcement and stopping the query process early. Corruptibility is defined as the number of input patterns making the output wrong under wrong keys [17] After every certain number of iterations in the SAT attack, random input patterns are utilized to query the activated chip, and the constraints from the queries are added to the CNF formula. If the output has low corruptibility, the portion of the input queries that generate the wrong output falls below a threshold. If this happens for a number of rounds, the algorithm terminates and returns an approximate key.

When the g and $\overline{g}$ in the Anti-SAT block of Fig. 1 are n-input AND and NAND gates, respectively, there is only one input pattern that makes the output wrong for each wrong key. Such an Anti-SAT design has low corruptibility and is subject to the AppSAT attack. To address this issue, it was proposed in [15] to combine the Anti-SAT block with traditional logic-locking schemes to increase the overall output corruptibility. However, the AppSAT attack can reduce this compound scheme to standalone SAT-resilient scheme, from which the keys can be recovered. The corruptibility has been increased in the SFLL [21], DTL [17], and ECE [22] schemes. Nevertheless, these designs lead to reduced number of iterations and hence query count in the SAT attack. There was a trade-off between the corruptibility and the query count in the previous designs.

D. Removal Attack

Removal attacks [18] can be utilized to identify the last gate of the logic-locking block, such as the gate G of the Anti-SAT block in Fig. 1. Then the output of this gate can be replaced by the correct signal to make the circuit function correct. For the type-0 and type-1 Anti-SAT blocks, the correct outputs are '0' and '1', respectively.

The removal attack for the Anti-SAT block can be carried out using the signal probability skew SPS, which is defined as

$$s_x = Pr[x = 1] - 0.5$$

for a signal x. Since $0 \leq Pr[x = 1] \leq 1$, the range of s_x is $[-0.5, 0.5]$. For a logic gate with two inputs whose SPS values are s_1 and s_2, its absolute difference (ADS) value is defined as

$$ADS = |s_1 - s_2|.$$

Assuming that X, K_1, K_2 are random. Then the SPS values of the inputs to the XOR gates in the Anti-SAT block are zero. Accordingly, the outputs of the XOR gates have zero SPS values. The SPS value for the output of an n-input AND gate is calculated as $s_{n-AND} = \prod_{i=1}^{n}(0.5 + s_i) - 0.5$, where s_i is the SPS of the i^{th} input. Since $s_i = 0$ for the AND gate in the g function, $s_{g(X,K_1)} = 0.5^n - 0.5$. As $n \to \infty$, $s_{g(X,K_1)} \approx -0.5$. Similarly, for the n-input NAND gate output from $\overline{g}$, the SPS is $s_{\overline{g}(X,K_2)} = 0.5 - 0.5^n$, which approaches 0.5 for large n. As a result, for the last gate, G, of the Anti-SAT block in Fig. 1(a), the output SPS is -0.5 and its ADS is $ADS_G = |s_{g(X,K_1)} - s_{\overline{g}(X,K_2)}| \approx 1$, if the number of inputs to the Anti-SAT block is large.

It was found in [18] that the ADS values for the gates in a circuit are rarely very high. Hence the G gate may be identified by first sorting out the gates with the highest ADS values. In the case that there are multiple candidates whose ADS values are very close, the transitive fan-in (TFI) of the candidate gates are analyzed. The TFI traces back the inputs of the candidate gates and finds how many key bits contribute to the inputs. The G gate should have all $2n$ key bits as contributors. Once the G gate is identified, its output signal can be replaced by '0' or '1' in the circuit when its SPS is negative or positive, respectively. This attack is named as the SPS attack.

In order to resist the SPS attack, structural obfuscation can be applied to the Anti-SAT scheme. After applying additional keys to obfuscate the structure, the SPS attack cannot detect the final gate G by simply sorting the ADS values. However, it was also mentioned in [18] that the removal and AppSAT attacks can be combined, which is termed as the AppSAT guided removal (AGR) attack. The combined attack uses the AppSAT attack to separate the keys for structural obfuscation from the keys to the Anti-SAT block. This is possible since the approximate values of the key inputs to the Anti-SAT block returned by the AppSAT attack over the iterations fluctuate due to the low corruptibility. After that, structural analyses can help the attacker find the final gate G. Then, similarly, the output signal of G is replaced by the correct value decided according to the SPS value.

III. Generalized Anti-SAT Constraints

The main reason that the Anti-SAT block is subject to the AppSAT and removal attacks is that p, the cardinality of the

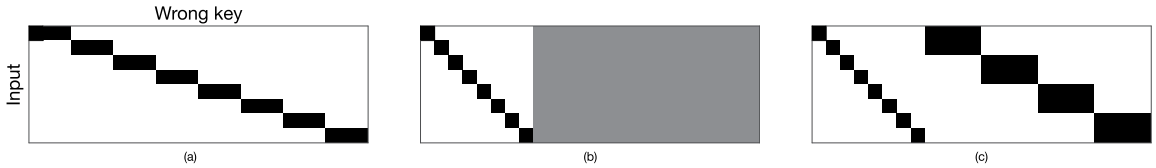

Fig. 2. Input-wrong key array, where a black block indicates wrong output for the corresponding input pattern and wrong key. (a) array for the Anti-SAT design; (b) array for the relaxed wrong key set constraint; (c) example array satisfying the relaxed wrong key set constraint.

TABLE I
SUMMARY OF NOTATIONS

L	The input vector to function f or g
X	The data input vector to the logic-locking block
λ	The number of iterations needed in the SAT attack
K_f	The key input vector to block f
$\mathbf{X}_n$	The set of all n-bit input vectors
$\mathbf{F}^T$	True set of function f
$\mathbf{F}^F$	False set of function f
$\mathbf{WK}_X$	The set of wrong keys that input X can exclude
$\mathbf{F}^F_{K_f}$	The set of all vectors in $\mathbf{F}^F$ XORed with K_f
$\mathbf{D}_{S-\mathbf{S}}$	$\{D\|D = S \oplus S_i, \forall S_i \in \mathbf{S}/S\}$.
$\mathbf{D}_\mathbf{S}$	$\{D\|D = S_1 \oplus S_2, \forall S_1 \neq S_2 \in \mathbf{S}\}$
$\oplus$	Bit-wise XOR operation
$\|\|$	Concatenation operation
$\&$	Logic AND operation
$+$	Logic OR operation

true set of the function g, is either too small or too big and the two functions are complement of each other. As a result, the corruptibility of the output is very low. On the other hand, such p is needed in the Anti-SAT design to maximize the number of iterations in the SAT attack. To solve this dilemma, true sets that have medium cardinality and at the same time lead to maximum SAT attack iterations are necessary. In this section, generalized constraints on the true sets for resisting the SAT attack are proposed. Our generalization allows a wide range of true set cardinality. Accordingly, logic-locking blocks with higher corruptibility can be designed to achieve better resilience to the AppSAT and removal attacks and resist the SAT attack at the same time. For convenience, some notations used in the following discussions are listed in Table I. In the remainder of this article, bold capital math symbols, such as $\mathbf{X}_n$ and $\mathbf{F}^F$, denote sets and capital symbols in normal font, such as X and F^F represent vectors.

A. Wrong Key Sets Analysis

Define $\mathbf{WK}_X$ as the wrong key set that input vector X can exclude. In other words, $\mathbf{WK}_X = \{WK | f_e(WK, X) \neq f_o(X)\}$. If the wrong key set of an input vector has already been covered by the wrong key sets of the DIPs in the previous iterations of the SAT attack, then this vector will not be selected as a DIP in the rest of the SAT attack. Therefore, if a circuit can be decrypted by the SAT attack in a limited number of iterations, there must be many input vectors whose wrong key sets cover each other and are not selected as DIPs.

Consider a circuit that has n inputs and requires λ iterations in the SAT attack. Denote the set of DIPs by $\mathbf{X}_{DIP}$. Accordingly, $\lambda = |\mathbf{X}_{DIP}|$. If a block needs to be resistant to the SAT attack, λ needs to be as big as possible, which is 2^n. This means that each possible n-bit input pattern can exclude some unique wrong keys that the other inputs cannot exclude. Take a 4-bit-input type-0 Anti-SAT block as an example. When p in (2) is 1, for each possible input X, $|\mathbf{WK}_X| = 15$. The key input has 8 bits and hence 2^8 different patterns. From [15], 16 of the keys are correct. Hence, the total number of wrong keys is $2^8 - 16 = 240$. Therefore, the number of DIPs and the number of iterations carried out by the SAT attack should be $\lambda \geq \frac{240}{15} = 16$. On the other hand, for 4-bit input, there are 2^4 patterns. Hence, $\lambda = 16$. Apparently, for this Anti-SAT scheme, the wrong key sets for different input patterns do not have any overlap. In other words,

$$\forall X_1 \neq X_2 \in \mathbf{X}_n, \quad \mathbf{WK}_{X_1} \cap \mathbf{WK}_{X_2} = \varnothing, \tag{3}$$

where $\mathbf{X}_n$ is the set of all possible input patterns of n bits. The wrong key sets of the Anti-SAT design can be illustrated by the array in Fig. 2 (a), in which the rows and columns represent all input patterns and wrong keys, respectively, and the black blocks in the row for input pattern X indicate $\mathbf{WK}_X$. It is clear in this figure that the wrong key sets for different input patterns do not overlap.

λ can still be made equal to 2^n to be resistant to the SAT attack even if there are overlaps among the wrong key sets. The Anti-SAT block is a special case. In addition, the functions of the two blocks do not have to be complementary of each other as in the Anti-SAT block.

B. Highlights of Proposed G-Anti-SAT Block

Our proposed G-Anti-SAT scheme generalizes the previous approach by allowing the wrong key sets of different input patterns to have overlaps. Instead of (3), our design requires that

$$\forall X_1 \neq X_2 \in \mathbf{X}_n, \exists K \; s.t. \; (K \in \mathbf{WK}_{X_1}) \& (K \notin \mathbf{WK}_{X_2}). \tag{4}$$

In other words, each wrong key set has at least one distinct element. This constraint is illustrated in the input-wrong key array in Fig. 2 (b). The distinct elements are denoted by the black cells in the diagonal. However, the rest of the input-wrong key array, as represented by the gray area, can have any patterns. Unlike that of the Anti-SAT design in Fig. 2 (a), the wrong key sets corresponding to different input patterns can have overlaps. Adopting the relaxation in (4), there are still $\lambda = 2^n$ DIPs. Hence, our generalized design is still resistant to the SAT attack. By allowing overlapping wrong key sets, the cardinality of the true set is relaxed so that it can be integers other than 1 or $2^n - 1$. The distinct element

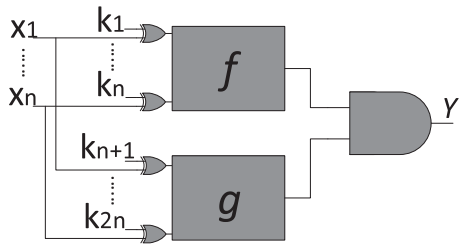

Fig. 3. Architecture of the proposed type-0 G-Anti-SAT block.

in each wrong key set is a wrong key that has only one input leading to the wrong output. However, by choosing functions with different true set cardinalities, the other wrong keys can have more inputs leading to the wrong output and hence higher corruptibility as shown by the example in Fig. 2 (c). As a result, better resiliency towards the AppSAT and removal attacks is achievable.

A second dimension of generalization is done in our design by allowing the two functions to be f and g, which are not necessarily complementary of each other. This enables larger design space and makes it even more difficult for the attacker to guess the functions adopted in the logic-locking block. Our non-complementary designs also have improved corruptibility and have better resilience towards the AppSAT and removal attacks.

In the following, subsection III.C.1) analyzes the constraints on the true sets to satisfy (4). When non-complementary blocks are adopted, it is non-trivial to identify the right keys. The constraint to ensure the existence of right keys is provided in subsection III.C.2). Section IV presents construction methods for the true sets by using K-maps. K-maps help to not only highlight the constraints need to be satisfied but also facilitate the design of the functions to achieve given true set cardinality.

C. Constraints for SAT Attack Resistance and Right Key Existence

1) Constraints for Resisting SAT Attack: Fig. 3 shows our proposed G-Anti-SAT block for type-0 design. Different from the Anti-SAT design [15], the functions of the two blocks do not have to be complementary of each other. Similarly, the last gate can be replaced by an OR gate to be a type-1 design. In the following, analysis is carried out on the type-0 design shown in Fig. 3. All the proposed analysis and constraints can be extended easily for the type-1 design.

Similarly, define

$$\mathbf{F}^T = \{L | f(L) = 1\} \quad \mathbf{F}^F = \{L | f(L) = 0\}.$$

Following the convention in [15], '1' is considered as the incorrect output for a type-0 block. For the architecture in Fig. 3, the output function is $y = f(X \oplus K_f) \& g(X \oplus K_g)$, where K_f and K_g are the key inputs of block f and g, respectively. $y =$ '1' only if $F^T = X \oplus K_f \in \mathbf{F}^T$ and $G^T = X \oplus K_g \in \mathbf{G}^T$. Therefore, the wrong key patterns in $\mathbf{WK}_X$ are in the format of $[X \oplus F^T || X \oplus G^T]$, where $||$ means concatenation. Accordingly, (4) can be interpreted as

$$\forall X_1 \neq X_2 \in \mathbf{X}_n, \exists F_1^T \neq F_2^T \in \mathbf{F}^T, G_1^T \neq G_2^T \in \mathbf{G}^T$$
$$s.t. \ [F_1^T \oplus X_1 \,||\, G_1^T \oplus X_1] \neq [F_2^T \oplus X_2 \,||\, G_2^T \oplus X_2]. \quad (5)$$

Since $\mathbf{X}_n$ includes all n-bit vectors, for any $F_1^T \neq F_2^T$, there must exists $X \in \mathbf{X}_n$ such that $F_1^T \oplus F_2^T = X$. X can be also rewritten as the sum of two elements in $\mathbf{X}_n$, *i.e.* $X = X_1 \oplus X_2$. Hence the constraint $\forall X_1 \neq X_2$, $F_1^T \oplus X_1 \neq F_2^T \oplus X_2$ can never be satisfied. Similarly, there do not exist $G_1^T \neq G_2^T$ such that $G_1^T \oplus X_1 \neq G_2^T \oplus X_2$, $\forall X_1 \neq X_2$. Therefore, to satisfy the constraints in (5), $\mathbf{F}^T$ and $\mathbf{G}^T$ need to be designed jointly so that $F_1^T \oplus X_1 = F_2^T \oplus X_2$ and $G_1^T \oplus X_1 = G_2^T \oplus X_2$ are not true at the same time.

Define the binary distance between two vectors X_1 and X_2 as $X_1 \oplus X_2$. Let $\mathbf{D}_{S-\mathbf{S}}$ be a set consisting of binary distances between an element $S \in \mathbf{S}$ and all the other elements in $\mathbf{S}$. In other words,

$$\mathbf{D}_{S-\mathbf{S}} = \{D | D = S \oplus S_i, \forall S_i \in \mathbf{S}/S\}. \quad (6)$$

Then $\mathbf{D}_{F^T-\mathbf{F}^T}$ is the set of vectors consisting of $F^T \oplus F_1^T$ for every $F_1^T \in \mathbf{F}^T$ and $F_1^T \neq F^T$. If $F^T \oplus X_1 = F_1^T \oplus X_2$ is satisfied, $F^T \oplus F_1^T = X_1 \oplus X_2$. Hence, $X_1 \oplus X_2$ is also in the set $\mathbf{D}_{F^T-\mathbf{F}^T}$. Similarly, the $X_1 \oplus X_2$ of the X_1 and X_2 satisfying the constraint that $G_1^T \oplus X_1 = G_2^T \oplus X_2$ is in the set $\mathbf{D}_{G^T-\mathbf{G}^T}$. Accordingly, the constraints in (5) are translated to

$$\textbf{Constraint 1: } \exists F^T \in \mathbf{F}^T, G^T \in \mathbf{G}^T$$
$$s.t. \, \mathbf{D}_{F^T-\mathbf{F}^T} \cap \mathbf{D}_{G^T-\mathbf{G}^T} = \varnothing. \quad (7)$$

The above equation gives a constraint equivalent to that in (4). However, this constraint can be utilized to construct $\mathbf{F}^T$ and $\mathbf{G}^T$ more easily.

From Constraint 1, it is clear that the functions f and g do not have to be complementary, and $|\mathbf{F}^T|$, $|\mathbf{G}^T|$ do not need to be 1 or $2^n - 1$. Therefore, the f and g blocks do not need to be AND and NAND gates, respectively, as in the Anti-SAT block [15]. The Anti-SAT block is just a special case of our proposed design. Many different functions can be chosen for f and g. In the design of f and g, an arbitrary set can be chosen as $\mathbf{F}^T$ first. For the selected $\mathbf{F}^T$, the choice of $\mathbf{G}^T$ may not be unique. Any $\mathbf{G}^T$ satisfying Constraint 1 can be utilized to achieve SAT-attack resilience.

Example 1: Take the structure in Fig. 3 with 4-bit input as an example. Different from the previous design, the f and g functions are allowed to be non-complementary. First, let $\mathbf{F}^T = \{0, 1, 2, 3\}$. To simplify the notations, decimal numbers are used to represent vectors here. For example, 2 represents the binary vector $[0, 0, 1, 0]$. It turns out $\{0, 8, 9, 11, 10\}$ is one of the possible sets for $\mathbf{G}^T$ that can satisfy Constraint 1. When $F^T = G^T = 0$, the two sets in Constraint 1 are disjoint. For a given $\mathbf{F}^T$, the corresponding $\mathbf{G}^T$, F^T, and G^T satisfying Constraint 1 can be found easily using K-maps and the procedure will be detailed in Section IV. $\mathbf{F}^T$ and $\mathbf{G}^T$ are the minterm numbers of the f and g functions, respectively. The logic formula for f and g can be derived accordingly. For the above choice of $\mathbf{F}^T$ and $\mathbf{G}^T$, $f(L) = \overline{l_3} \& \overline{l_2}$ and $g(L) = l_3 \& \overline{l_2} + \overline{l_2} \& \overline{l_1} \& \overline{l_0}$, where $L = [l_3, l_2, l_1, l_0]$ is the 4-bit input

to f and g and '&' and '+' denote the logic AND and OR operations, respectively.

Example 2: The f and g in our design can be complementary as well. Select $\mathbf{F}^T = \{6, 8, 9, 10, 11, 12, 13, 14, 15\}$ and $\mathbf{G}^T = \{0, 1, 2, 3, 4, 5, 7\}$. It can be found that $F^T = 5$ and $G^T = 6$ satisfy Constraint 1. The corresponding functions f and g are $f(L) = l_3 + l_2 \& l_1 \& \overline{l_0}$ and $g(L) = \overline{f(L)}$.

Example 3: Constraint 1 is not sufficient to guarantee the existence of right keys. For example, take $\mathbf{F}^T = \{0, 1, 2, 3\}$ and $\mathbf{G}^T = \{0, 4, 8, 12\}$. Constraint 1 is also satisfied by taking $F^T = G^T = 0$. Accordingly, $f(L) = \overline{l_3} \& \overline{l_2}$ and $g(L) = \overline{l_1} \& \overline{l_0}$. However, in this case, from exhaustive search, there does not exist a right key K^* such that $f_e(X, K^*) = f_o(X)$ for every possible X.

2) Constraints for Existence of Right Keys: When f and g are not complementary, additional constraints need to be introduced to guarantee the existence of right keys.

The correct output of the type-0 logic-locking block in Fig. 3 is '0'. Hence, right keys are $[K_f \,||\, K_g]$ such that for every $X \in \mathbf{X}_n$, $f(X \oplus K_f) = 0$ or $g(X \oplus K_g) = 0$. Define $\mathbf{F}^F_{K_f} = \{X | X = F^F \oplus K_f,\ \forall F^F \in \mathbf{F}^F\}$. It is the set of X that makes $f(X \oplus K_f) = 0$. Similarly, $\mathbf{G}^F_{K_g} = \{X | X = G^F \oplus K_g,\ \forall G^F \in \mathbf{G}^F\}$ is the set of X that makes $g(X \oplus K_g) = 0$. Therefore, the right keys $[K_f \,||\, K_g]$ should satisfy

$$(\mathbf{F}^F_{K_f} \cup \mathbf{G}^F_{K_g}) = \mathbf{X}_n. \tag{8}$$

For a selected function f, a function g can be designed to satisfy (8). From the definition, $\mathbf{F}^T \cup \mathbf{F}^F = \mathbf{X}_n$ and $\mathbf{F}^T \cap \mathbf{F}^F = \varnothing$. Hence, for any K_f, $\mathbf{F}^F_{K_f} \cup \mathbf{F}^T_{K_f} = \mathbf{X}_n$. In the case that f and g are not complementary, if $\mathbf{G}^F_{K_g} \supseteq \mathbf{F}^T_{K_f}$, then (8) would be satisfied. Define the binary distance structure of a set $\mathbf{S}$ as

$$\mathbf{D_S} = \{D | D = S_1 \oplus S_2, \forall S_1 \neq S_2 \in \mathbf{S}\}. \tag{9}$$

It should be noted that the binary distance structure may have repeated elements and the order of the elements does not matter. If two binary distance structures have the same elements and the numbers of each element are the same, then they are considered as the same binary distance structure. It was found that to make $\mathbf{G}^F_{K_g} \supseteq \mathbf{F}^T_{K_f}$, $\mathbf{G}^F$ should have a subset with the same binary distance structure as $\mathbf{F}^T$. In other words,

$$\textbf{Constraint 2:} \quad \exists \mathbf{S} \subset \mathbf{G}^F,\ s.t.\ \mathbf{D_S} = \mathbf{D}_{\mathbf{F}^T}$$

The proof is detailed in the appendix.

Let us use Constraint 2 to check whether right keys exist for Examples 1 and 3 in the last subsection.

1) In Example 1, $\mathbf{F}^T = \{0, 1, 2, 3\}$ and $\mathbf{G}^T = \{0, 8, 9, 11, 10\}$. From (9), it can be computed that $\mathbf{D}_{\mathbf{F}^T} = [1, 2, 3, 3, 2, 1, 1, 2, 3, 3, 2, 1]$. A subset $\mathbf{S}$ of $\mathbf{G}^F$ that has the same binary structure as $\mathbf{F}^T$ is $\{12, 13, 14, 15\}$. $\mathbf{D_S} = [1, 2, 3, 3, 2, 1, 1, 2, 3, 3, 2, 1]$. Hence, this block has right keys. The method to find the right keys will be presented in Section IV. It can be found that one of the right keys is $K_f = [0, 0, 0, 0]$ and $K_g = [0, 0, 0, 1]$

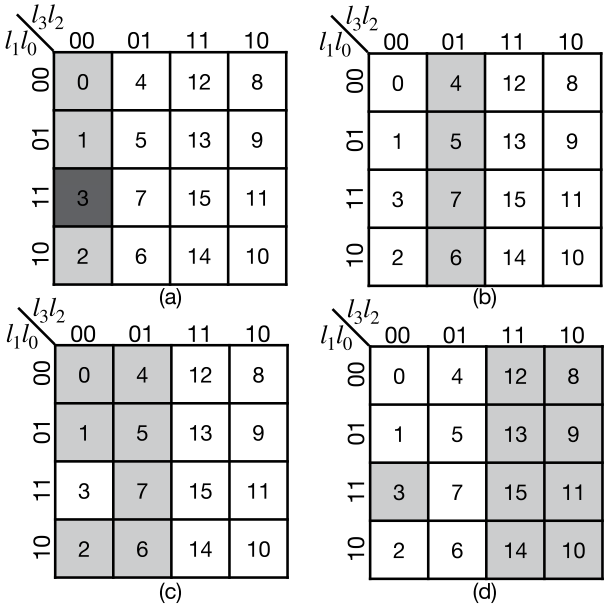

Fig. 4. (a) Randomly selected cells forming $\mathbf{F}^T$, and the darker gray cell is the common cell shared with $\mathbf{G}^T$; (b) One group of cells with the same binary distance structure as those in (a); (c) Cells that should be covered by $\mathbf{G}^F$ at least; (d) Cells for corresponding $\mathbf{G}^T$.

2) In Example 3, $\mathbf{F}^T = \{0, 1, 2, 3\}$ and hence $\mathbf{D}_{\mathbf{F}^T} = [1, 2, 3, 3, 2, 1, 1, 2, 3, 3, 2, 1]$. However, there is no subset of $\mathbf{D}_{\mathbf{G}^F}$ having the same binary structure as $\mathbf{F}^T$. Hence right key does not exist.

The proposed constraints for type-0 blocks can be easily extended to design type-1 G-Anti-SAT blocks. For type-1 blocks, Constraints 1 and 2 should be modified as

$$\exists F^F \in \mathbf{F}^F, G^F \in \mathbf{G}^F\ s.t.\ \mathbf{D}_{F^F - \mathbf{F}^F} \cap \mathbf{D}_{G^F - \mathbf{G}^F} = \varnothing,$$
$$\exists \mathbf{S} \subset \mathbf{G}^T\ s.t.\ \mathbf{D_S} = \mathbf{D}_{\mathbf{F}^F}$$

IV. Generalized Anti-SAT Block Design Using K-Maps

This section proposes methods for designing the true sets for type-0 blocks that satisfy Constraint 1, 2 and finding right keys. The proposed methods are developed using K-maps. The elements in the true sets are mapped to the cells in the K-map. Accordingly, designing the true sets is translated to grouping the cells in the K-map. Whether the constraints are satisfied can be easily observed from the K-map. Also K-maps help to design blocks with lower logic complexity. The proposed design approaches can be extended similarly for type-1 blocks.

A. K-Map Cell Selection for Non-Complementary Functions

Let us first focus on the case that the functions f and g are non-complementary. Consider the G-Anti-SAT block in Fig. 3 with 4-bit input $L = [l_3, l_2, l_1, l_0]$ to f and g as an example. The corresponding K-map has 16 cells represented as a 4×4 array. l_3l_2 and l_1l_0 are used to label the columns and rows, respectively, as shown in Fig. 4. The minterm numbers of the cells in the K-map are also listed in the figure.

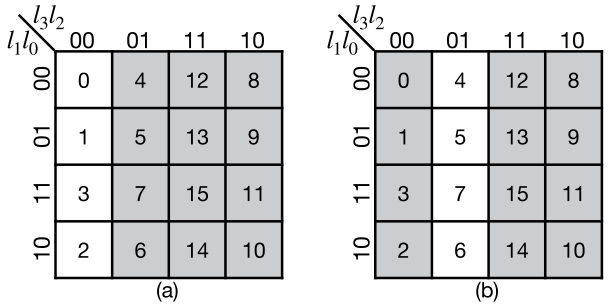

Fig. 5. (a) cells of $\mathbf{F}^F$; (b) cells of $\mathbf{F}^F_{K_f}$ with $K_f = [0100]$.

Constraint 1 requires that there exist $F^T \in \mathbf{F}^T$ and $G^T \in \mathbf{G}^T$ satisfying $\mathbf{D}_{F^T-\mathbf{F}^T} \cap \mathbf{D}_{G^T-\mathbf{G}^T} = \varnothing$. When $\mathbf{F}^T$ and $\mathbf{G}^T$ are non-complementary, the groups of cells for $\mathbf{F}^T$ and $\mathbf{G}^T$ can have overlaps in the K-map and a common cell can be used as both F^T and G^T. In a K-map, the column and row labels for each cell are distinct. Hence, adding the label of a cell to the labels of each of the other cells leads to a set of distinct labels. Accordingly, a random cell, such as the dark gray one in Fig. 4(a), can be chosen as the cell representing both F^T and G^T. Then $\mathbf{F}^T$ and $\mathbf{G}^T$ can be formed by including non-overlapping cells among the remaining cells. Also $\mathbf{F}^T$ and $\mathbf{G}^T$ do not need to cover all the cells.

When f and g are non-complementary, additional constraints need to be added to the K-map cell selection in order to satisfy Constraint 2 and hence have right keys. Let us include a quarter of the cells in the K-map for $\mathbf{F}^T$. For the common cell shown in Fig. 4(a), the cells in the leftmost column of the K-map can be randomly selected to be $\mathbf{F}^T$. Although any cells including the common cell can be chosen, using adjacent cells leads to reduced logic complexity. Three groups of cells in the K-map have the same binary distance structure defined in (6) as $\mathbf{F}^T$. One example is the group of cells shown in Fig. 4(b). Having $\mathbf{G}^F$ include at least the cells in Fig. 4(b) would satisfy Constraint 2. On the other hand, $\mathbf{G}^T$ covers all the other cells not covered by $\mathbf{G}^F$ and can only share one common cell with $\mathbf{F}^T$. Therefore, $\mathbf{G}^F$ also needs to cover every cell in $\mathbf{F}^T$ except the common cell. For the $\mathbf{F}^T$ selected in Fig. 4(a), Fig. 4(c) shows the $\mathbf{G}^F$ satisfying these requirements. The remaining cells, as shown in Fig. 4(d), form $\mathbf{G}^T$ satisfying Constraint 1 and 2. It should be noted that there exist other choices of $\mathbf{F}^T$ and $\mathbf{G}^T$ satisfying Constraints 1 and 2 besides the ones that can be found by using the above method.

The right keys K_f and K_g can be easily decided from the cells for $\mathbf{F}^F$ and $\mathbf{G}^F$ in the K-map. Constraint 2 is equivalent to (8). In the K-map, the group of cells for $\mathbf{F}^F_{K_f}$ has the same shape as that for $\mathbf{F}^F$, except that it is shifted and/or flipped according to the K_f vector. Similarly, the group of cells for $\mathbf{G}^F_{K_g}$ is that for $\mathbf{G}^F$ shifted and/or flipped according to K_g. Then (8) is translated to that the shifted and/or flipped groups for $\mathbf{F}^F$ and $\mathbf{G}^F$ need to cover every cell in the K-map. The vectors leading to such shifting/flipping are the right keys K_f and K_g. For the $\mathbf{F}^T$ in Fig. 4(a), Fig. 5(a) shows the cells for the corresponding $\mathbf{F}^F$. The cells for $\mathbf{G}^F$ are illustrated in Fig. 4(c). It can be seen that the union of such $\mathbf{G}^F$ and $\mathbf{F}^F$ covers every cell in the K-map except the one with $[l_3, l_2, l_1, l_0] = [0, 0, 1, 1]$, which is the common cell. One way to cover every cell in the K-map is to keep the cells for $\mathbf{G}^F$ unchanged, which means $K_g = [0, 0, 0, 0]$, and use $K_f = [0, 1, 0, 0]$, which leads to the group of cells of $\mathbf{F}^F_{K_f}$ shown in Fig. 5(b). The gray cells in Fig. 4(c) and Fig. 5(b) are $\mathbf{G}^F_{K_g}$ with $K_g = [0, 0, 0, 0]$ and $\mathbf{F}^F_{K_f}$ with $K_f = [0, 1, 0, 0]$, respectively. They cover all cells in the K-map. There are many choices of K_f and K_g that satisfy (8). Another example is $K_g = [0, 1, 0, 0]$ and $K_f = [0, 0, 0, 0]$. It corresponds to that the $\mathbf{F}^F$ in Fig. 5(a) is unchanged and the column labels of the cells in $\mathbf{G}^F$ are XORed with the 2-bit vector [0,1] to form $\mathbf{G}^F_{K_g}$. Hence, in K-map, the cells for $\mathbf{G}^F_{K_g}$ are those in Fig. 4(c) flipped horizontally.

Our proposed design can be easily generalized to n-bit input $L = [l_{n-1}, l_{n-2}, \cdots, l_1, l_0]$. The cardinality of $\mathbf{F}^T$ is chosen to be 2^{n-t} $(2 \le t \le n-1)$ to simplify the logic. To facilitate the design, the K-map can be drawn as an array of $2^{n-t} \times 2^t$ cells. Use $l_{n-t-1}, \cdots l_1, l_0$ to label the rows and $l_{n-1}, \cdots l_{n-t+1}, l_{n-t}$ to label the columns. Then $\mathbf{F}^T$ can cover a column of the K-map, and the f function is the AND operation among the t bits of the column label. To satisfy Constraint 1, $\mathbf{G}^T$ should include exactly one element from $\mathbf{F}^T$. Besides, to satisfy Constraint 2, $\mathbf{G}^T$ can include the remaining columns except one whose column label is different in one single bit from the column label of $\mathbf{F}^T$ as in the example presented in Fig. 4(d). Accordingly, $\mathbf{G}^T$ can include $2^t - 2$ columns plus one cell in the K-map and $|\mathbf{G}^T| = 2^n - 2^{n-t+1} + 1$. Without loss of generality, let $\mathbf{F}^T$ cover the column with all '0' label. Then

$$f(L) = \bar{l}_{n-t} \& \bar{l}_{n-t+1} \& \cdots \& \bar{l}_{n-1}. \tag{10}$$

Assume that the two columns not in $\mathbf{G}^T$ are different in bit l_q in their labels. Then the logic expression for the part of $\mathbf{G}^T$ that consists of the $2^t - 2$ columns is

$$g_1(L) = l_{n-t} + \cdots + l_{q-1} + l_{q+1} + \cdots + l_{n-1}. \tag{11}$$

The one cell from $\mathbf{F}^T$ that is also included in $\mathbf{G}^T$ can be grouped with the other cells in $\mathbf{G}^T$ to simplify the logic expression. If the cell whose row label is all '0' is picked as this common cell, then the logic expression covering this cell is

$$g_2(L) = \bar{l}_q \& \bar{l}_0 \& \bar{l}_1 \& \cdots \& \bar{l}_{n-t-1}. \tag{12}$$

Overall, $g(L) = g_1(L) + g_2(L)$. Of course, a different column can be chosen for $\mathbf{F}^T$ and an alternative common cell can be used. In this case, the literals in the $f(L)$ and $g(L)$ functions need to be complemented accordingly. In order to increase the corruptibility of the output signal, the product of $|\mathbf{F}^T|$ and $|\mathbf{G}^T|$ needs to be as large as possible. The best design for achieving this goal for the n-bit non-complementary design we found so far is to let $|\mathbf{F}^T| = 2^{n-2}$, $|\mathbf{G}^T| = 2^{n-1} + 1$.

The right keys for the above non-complimentary design can be also obtained by flipping/shifting the false sets of f or g to make their union cover all the cells in the n-bit K-map. Equivalently, this means that there should be no overlap between $\mathbf{G}^T_{K_g}$ and $\mathbf{F}^T_{K_f}$. To meet this requirement, the right key can be any $K_f = [K_{f,n-1}, K_{f,n-2}, \cdots, K_{f,0}]$

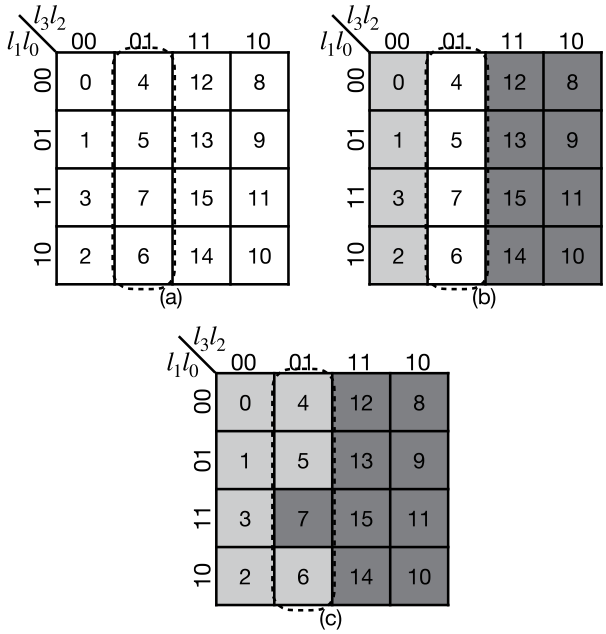

Fig. 6. (a) The column circled in dashed line is the dividing column; (b) Splitting the other columns between $\mathbf{F}^T$ (light gray) and $\mathbf{G}^T$ (dark gray); (c) Splitting the cells in the dividing column between $\mathbf{F}^T$ and $\mathbf{G}^T$.

and $K_g = [K_{g,n-1}, K_{g,n-2}, \cdots, K_{g,0}]$ such that $K_{f,i} = K_{g,i}$ $(n-t \leq i \leq n-1, i \neq q)$ and $K_{f,q} = \overline{K_{g,q}}$. The bits in the other positions can be either '0' or '1'.

In summary, for a chosen cardinality $|\mathbf{F}^T| = 2^{n-t}$ $(2 \leq t \leq n-1)$, an n-bit non-complementary G-Anti-SAT block can be designed according to the following steps.

1) Select a column in the $2^{n-t} \times 2^t$ K-map for $\mathbf{F}^T$; Select one cell in this column as the common cell; Select an integer q in the range of $[n-t, n-1]$;
2) Design $f(L)$ and $g(L) = g_1(L) + g_2(L)$ using (10), (11) and (12). If a bit in the labels for the column and common cell selected in Step 1) is '0', then use the literals as in (10), (11) and (12). If the bit is '1', complement the corresponding literals in the equations.
3) Pick any K_f and K_g such that they are different in bit q and are the same in bits $n-t$ through $n-1$ to be a right key.

It should be noted that $\mathbf{F}^T$ and $\mathbf{G}^T$ do not have to cover entire columns as in the design process explained above. This would provide more variation on the design at the cost of higher logic complexity.

B. K-Map Cell Selection for Complementary Functions

When f and g are complementary, $\mathbf{F}^T$ and $\mathbf{G}^T$ should not have any common cells and should cover all the cells in the K-map. First pick a random column as shown in Fig. 6(a). This column is referred to as the dividing column in this article. The labels for each column in the K-map are distinct. Hence, adding the column label of the dividing column to the labels of the other columns results in a set of distinct nonzero vectors. This means that splitting the other columns between $\mathbf{F}^T$, $\mathbf{G}^T$ and picking F^T, G^T from the dividing column would satisfy Constraint 1. For example, in Fig. 6(b), the first column is put in $\mathbf{F}^T$ and the third and fourth columns are put in $\mathbf{G}^T$.

Next, the cells in the dividing column should be split between $\mathbf{F}^T$ and $\mathbf{G}^T$. Put one cell of this column in one set and the others in the other set. Without loss of generality, the one cell is put in $\mathbf{G}^T$ and the other cells are put in $\mathbf{F}^T$, as shown by the example in Fig. 6(c). The cell of $\mathbf{G}^T$ in the dividing column can be used as G^T and any other cells in the dividing column can be used as F^T. The column labels of any two cells in the same column are the same. Hence, adding F^T to any other cells in the dividing column would result in a zero column label, and any vector in $\mathbf{D}_{G^T - \mathbf{G}^T}$ is different from those in $\mathbf{D}_{F^T - \mathbf{F}^T}$. As a result, splitting $\mathbf{F}^T$ and $\mathbf{G}^T$ in this way satisfies Constraint 1.

As mentioned previously, when f and g are complementary, any $K_f = K_g$ can be used as a right key. Constraint 2 does not need to be considered in this case.

The complementary G-Anti-SAT design can be also easily generalized to n-bit input $L = [l_{n-1}, l_{n-2}, \cdots, l_1, l_0]$. The cardinality of $\mathbf{F}^T$ is chosen to be in the format of $2^{n-t} - 1$ $(1 \leq t \leq n-1)$. Then $|\mathbf{G}^T| = 2^n - 2^{n-t} + 1$. To facilitate the design, the K-map is drawn as an array of $2^{n-t} \times 2^t$ cells and the columns and rows are labeled in the same way as in the non-complementary design. As in the example of Fig. 6(a), a column is first selected as the dividing column. To satisfy Constraint 1, one cell in the dividing column and the remaining columns are put in $\mathbf{G}^T$, and the remaining cells in the dividing column are included in $\mathbf{F}^T$. Without loss of generality, choose the column with all '0' label as the dividing column. Then the logic expression for the part of $\mathbf{G}^T$ that consists of the remaining $2^t - 1$ columns is

$$g_1(L) = l_{n-t} + l_{n-t+1} + \cdots + l_{n-1}. \tag{13}$$

The cell in the dividing column that is included in $\mathbf{G}^T$ can be grouped with the other cells in the same row of the K-map that are also in $\mathbf{G}^T$. In the case that this cell has all '0' in its label, the logical expression covering this cell is

$$g_2(L) = \bar{l}_0 \& \bar{l}_1 \& \cdots \& \bar{l}_{n-t-1} \tag{14}$$

Overall, $g(L) = g_1(L) + g_2(L)$, $f(L) = \overline{g(L)}$. A different column can be chosen as the dividing column and an alternative cell can be put in $\mathbf{G}^T$. In this case, the literals in $g_1(L)$ and $g_2(L)$ formulas need to be complemented accordingly. For the n-bit complementary design, the maximum corruptibility in the output is achieved when $|\mathbf{F}^T| = 2^{n-1} - 1$ and $|\mathbf{G}^T| = 2^{n-1} + 1$.

In summary, for a chosen cardinality $|\mathbf{F}^T| = 2^{n-t} - 1$ $(1 \leq t \leq n-1)$, an n-bit complementary G-Anti-SAT block can be designed according to the following steps.

1) Select a column of the $2^{n-t} \times 2^t$ K-map as the dividing column; Select one cell in this column to put in $\mathbf{G}^T$
2) Design $g(L) = g_1(L) + g_2(L)$ using (13) and (14). Then $f(L) = \overline{g(L)}$. If a bit in the labels for the dividing column and the cell selected in Step 1) is '0', then use the literals as in (13) and (14). If the bit is '1', complement the corresponding literals in the equations.
3) Any $K_f = K_g$ can be used as the right key.

Similarly, $\mathbf{F}^T$ and $\mathbf{G}^T$ do not have to cover entire columns as in the design process explained above. Other designs satisfying

TABLE II

NUMBER OF ITERATIONS AND TIME NEEDED BY THE SAT ATTACK TO DECRYPT THE G-ANTI-SAT AND ANTI-SAT BLOCKS

		$n=8$	$n=12$	$n=16$
non-complementary G-Anti-SAT	# of iterations	255	4095	-
	time (second)	0.44	66.21	timeout
complementary G-Anti-SAT	# of iterations	255	4095	-
	time (second)	0.80	166.42	timeout
Anti-SAT block [15]	# of iterations	255	4095	-
	time (second)	0.82	175.74	timeout

TABLE III

CORRUPTIBILITIES (e) OF WRONG KEYS AND AVERAGE CORRUPTIBILITY FOR n-BIT G-ANTI-SAT DESIGNS

	# of wrong keys	**Corruptibility** (e)	**Average corruptibility**
Complementary G-Anti-SAT	$2^{2n}-2^{2n-t}$	$2^{n-t}-1$	$2^{n-t}-2^{n-2t}$
	$2^{2n-t}-2^{n}$	1	
Non-complementary G-Anti-SAT	$2^{2n}-2^{2n-t+1}$	2^{n-t}	$2^{n-t}-2^{n-2t+1}$
	2^{2n-t}	1	

Constraint 1 are possible, although the logic complexity may be higher

V. EXPERIMENTS, ANALYSIS, AND COMPARISONS

This section evaluates the effectiveness of our proposed G-Anti-SAT scheme against the SAT, AppSAT, and removal attacks. The SAT attack tool in [14] based on the Lingeling SAT solver is used in our experiments to test the running time of the SAT attack. The CPU time is limited to 10 hours, and the experiments are run over an Intel Core i7 with 4GB RAM. The resiliency of our logic-locking blocks against the AppSAT attack is evaluated by analyzing the average corruptibility and the corruptibility profile of the approximate keys that can be returned by the attack. This analysis is enabled by the tool in [26] that can return an approximate key after any given number of SAT attack iterations. The evaluation for the removal attack resistance is done by calculating the ADS values as in [18]. At the end, the area requirement of our design from synthesis reports is compared to that of prior designs.

A. SAT Attack Resistance Analysis

The SAT attack is applied to decrypt the proposed G-Anti-SAT blocks, and the number of iterations and time are listed in Table II for the designs with different numbers of input bits. For comparison, the Anti-SAT block is also simulated in the same hardware environment and the results are included in Table II. It can be observed from this table that our design achieves the same resistance to the SAT attack in terms of the number of iterations, which matches our previous analysis. The reason that the time consumed by the attack on the non-complementary G-Anti-SAT block is shorter than those on the complementary G-Anti-SAT and Anti-SAT blocks is because that the non-complementary design has less complicated logic. As a result, the complexity to construct and solve the corresponding CNF formula is lower. From the table, we can see that for 16-bit input, the time needed by the SAT attack to decipher the G-Anti-SAT block is already over 10 hours. Also the time needed for the attack increases very fast with the input size.

B. AppSAT Attack Resistance Analysis

The corruptibility of each wrong key, which is the number of input patterns leading to the wrong output, is 1 in the Anti-SAT scheme [15]. Adopting the relaxation on the wrong key sets, our proposed G-Anti-SAT scheme can increase the corruptibility of a large portion of the wrong keys. Hence, in each of the two proposed schemes, the wrong keys have two possible corruptibilities. The corruptibilities denoted by e and the number of wrong keys with those corruptibilities are listed in Table III for each of the proposed G-Anti-SAT schemes with n-bit input and parameter t. The average corruptibility is computed as the sum of the corruptibility of each wrong key divided by the total number of wrong keys. The average corruptibility is maximized in the complementary design when $t=1$, and it is around 2^{n-2}.

The average corruptibility is not the single criterion to evaluate the resiliency to the AppSAT attack. It is possible that a large average corruptibility is resulted from a small portion of wrong keys with very high corruptibility. In this case, those high-corruptibility keys will be excluded by the AppSAT attack in a few iterations, and the key returned by the AppSAT attack will have low corruptibility. As listed in Table III, each wrong key in our G-Anti-SAT design can have one of the two possible corruptibilities, which are referred to as the high corruptibility and low corruptibility in the remainder of the discussion. In our design, a larger t leads to lower average corruptibility but a larger portion of wrong keys with high corruptibility. As a result, when the AppSAT attack is applied, the chance of returning a high-corruptibility key is larger, although the high corruptibility for the design with t is smaller than that of the design with $t' < t$.

The running time of the AppSAT attack depends on many parameters, such as the number of random patterns for each round, error threshold, and settlement count [17]. Besides, these parameters can be adjusted according to the logic-locking scheme. Hence, the running time of the AppSAT attack is not a good measurement of the resiliency towards this attack. Instead, the corruptibility of the key returned by the AppSAT attack affects the usability of the key a lot. Hence, in the following, the focus is given to the corruptibilities of the keys that can be returned after different numbers of SAT iterations.

The corruptibility profiles of 25-bit-input G-Anti-SAT designs with different t are presented in Fig. 7. In these plots, the x-axis is the iteration number in the SAT attack. A key is returned after every 50 iterations. The y-axis is the corruptibility of the returned key. For $t=3$, the high corruptibility is very large, which is $2^{25-3}=2^{22}$ and $2^{22}-1$ for the non-complementary and complementary designs, respectively, from Table III. However, the portion of the wrong keys with this high corruptibility is very small. As a result, as shown by the two plots in the first row of Fig. 7, the chance of returning a low-corruptibility key is very high. As t increases

TABLE IV
CORRUPTIBILITY FOR n-BIT ANTI-SAT-DTL SCHEME

Gate-type	Corruptibility (e)	Average corruptibility
XOR	$[(2(2^{2^{l-1}}-1))^{r-1}, (2(2^{2^{l-1}}-1))^{r}]$	$(2(2^{2^{l-1}}-1))^r(2^n-(2(2^{2^{l-1}}-1))^r)/2^n$
OR/NAND	$[(1+2(2^{2^{l-1}}-1))^{r-1}, (1+2(2^{2^{l-1}}-1))^{r}]$	$(1+2(2^{2^{l-1}}-1))^r(2^n-(1+2(2^{2^{l-1}}-1))^r)/2^n$

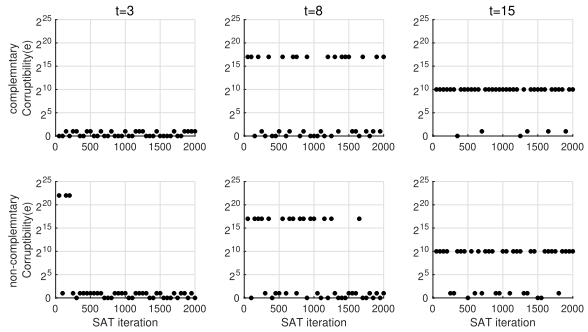

Fig. 7. Corruptibility of the returned keys over SAT attack iterations for G-Anti-SAT blocks with $n = 25$.

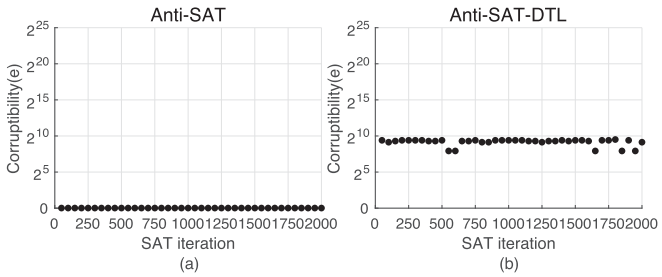

Fig. 8. Corruptibilities of the returned keys over SAT attack iterations for logic-locking blocks with $n = 25$. (a) Anti-SAT; (b) Anti-SAT-DTL with 6 OR gates in the first layer.

to 8 and 15, the high corruptibility is reduced to around 2^{17} and 2^{10}, respectively. Nevertheless, the portion of the wrong keys with high corruptibility increases exponentially with t. Hence, as it can be observed from the plots in the second and third columns of Fig. 7, the chance of returning a high-corruptibility key becomes much higher for larger t. It should be noted that the value of the high corruptibility decreases for larger t. When $t = n - 1$, the value of the high corruptibility reduces to 1. Hence using a large t close to $n - 1$ does not lead to better resiliency to the AppSAT attack, and a medium t strikes a balance between the value of the corruptibility and the chance of returning a wrong key with good corruptibility.

For reference, the corruptibility profiles of the Anti-SAT and Anti-SAT-DTL scheme [17] with $n = 25$-bit input are shown in Fig. 8. As expected, for the Anti-SAT design, the corruptibility of the wrong key returned is always 1. In the Anti-SAT-DTL scheme, the AND/NAND trees in the Anti-SAT block are replaced by n-bit DTL. When r AND gates in layer l ($1 \leq l \leq \lceil log_2 n \rceil$) are replaced by XOR/OR/NAND gates, it has been derived in [17] that the corruptibilities of the wrong keys are in the range shown in Table IV. The average corruptibility can be also calculated as listed in this table. Fig. 8(b) is for the case that 6 AND gates in the first layer are replaced by OR gates. Since the corruptibilities, e, of all wrong keys are high in the Anti-SAT-DTL design, it is more resilient to the AppSAT attack compared to our G-Anti-SAT scheme. However, its better AppSAT attack resiliency comes at the cost of reducing the number of iterations needed by the SAT attack to $2^n/e$ [17].

C. Removal Attack Resistance Analysis

The SPS removal attack utilizes the ADS values. The ADS value of a gate is in the range of [0,1). However, it is rare that a circuit contains a gate whose ADS value is very close to 1 as the last gate of the Anti-SAT block. Hence, the last gate can be identified by sorting the ADS values. The ADS values of most gates in a circuit are in a medium range. By tuning the parameter t, the cardinalities of the true sets of the f and g functions of our G-Anti-SAT design change. Accordingly, the ADS value of the last gate varies and it is well-hidden among the ADS values of the other gates. As a result, unlike the Anti-SAT design, it is difficult to identify the last gate in our G-Anti-SAT design by finding the gate with either the largest or smallest ADS value. For example, a non-complimentary G-Anti-SAT block can be designed to have 2^{n-2} and $2^{n-1}+1$ as the cardinalities of $\mathbf{F}^T$ and $\mathbf{G}^T$, respectively. In this case, the ADS value of the last gate is $|\frac{2^{n-2}}{2^n} - \frac{2^{n-1}+1}{2^n}| \approx 0.25$.

In order to resist the SPS attack, structural and functional obfuscations can be inserted into the Anti-SAT block [15]. However, the keys for structural and functional obfuscation can be separated from those for the Anti-SAT block by the AGR attack [18]. After that, the last gate and the correct output signal can still be obtained. The reason that the keys can be separated is because that the keys to the Anti-SAT scheme has low corruptibility while those for the obfuscation schemes have high corruptibility. In our G-Anti-SAT designs, the output already has higher corruptibility even if a larger t is used to achieve better AppSAT resistance. Hence, if our design is combined with structural and functional obfuscations, the keys are harder to be separated by using the AGR attack and even better resistance to the removal attack can be achieved.

D. Complexity Comparison

Our proposed complementary and non-complementary G-Anti-SAT designs with different n and $t = 3$ are synthesized using TSMC 65nm CMOS process under 4ns timing constraints, and the results are listed in Table V. For comparison, the area of the Anti-SAT design is also included in this table. Our complementary design has very similar area as the Anti-SAT design. On the other hand, our non-complementary design is much smaller. The reason is that the true set of one function in the non-complementary design consists of a whole column of the K-map and hence its logic expression is substantially simpler. To show how the complexity of our

TABLE V

AREAS OF LOGIC-LOCKING BLOCKS WITH $t = 3$ SYNTHESIZED USING TSMC 65*nm* PROCESS WITH 4*ns* TIMING CONSTRAINT

	$n = 8$	$n = 16$	$n = 25$
non-complementary G-Anti-SAT(μm^2)	91.440	153.000	222.480
complementary G-Anti-SAT(μm^2)	131.400	254.880	393.480
Anti-SAT (μm^2)	129.600	253.080	393.120

TABLE VI

AREAS OF G-ANTI-SAT BLOCKS WITH $n = 25$ SYNTHESIZED USING TSMC 65*nm* PROCESS WITH 4*ns* TIMING CONSTRAINT

	$t = 3$	$t = 8$	$t = 15$
non-complementary G-Anti-SAT(μm^2)	222.480	261.000	316.800
complementary G-Anti-SAT(μm^2)	393.480	394.200	392.760

designs change with t, synthesis results for $n = 25$ and different t are listed in Table VI. It can be observed that, when larger t is adopted, the area for the non-complementary case increases. This is because more literals are included in the f function as shown in (10). On the other hand, the area requirement for the complementary case remains similar for different t since changing t only leads to switching literals between the $g_1(L)$ and $g_2(L)$ functions in (13) and (14).

VI. DISCUSSIONS

Our proposed G-Anti-SAT designs enjoy great flexibility on the f and g functions. They do not have to be AND/NAND tree or complementary to make the query count needed in the SAT attack exponential. Unlike previous designs, the true sets of the functions can be chosen to have larger cardinalities and hence increase the corruptibility without sacrificing the SAT-attack resilience.

There is another attack called the bypass attack [27]. The main idea is to add a bypass circuit that inverts the wrong output and nullifies the effect of the wrong key on the encrypted circuit. The complexity of the bypass circuit increases linearly with N_p, the number of input patterns that generate the wrong output. In the Anti-SAT design, each wrong key only leads to wrong output for one and only one input pattern and hence $N_p = 1$. However, for an n-bit complementary G-Anti-SAT design with $|\mathbf{G}^T| = 2^{n-t} - 1$, there are $2^{2n} - 2^{2n-t}$ wrong keys with $2^{n-t} - 1$ as the corruptibility from Section V.B. If any of those wrong keys is used, $N_p = 2^{n-t} - 1$ and hence the overhead of the corresponding bypass circuit is much higher. Similarly, to apply the bypass attack on the non-complementary G-Anti-SAT design, the area overhead would be also very high due to the high corruptibility of the wrong keys.

It was also realized that the wrong key sets may have overlaps in the recent CAS-Lock design [25]. However, unlike our G-Anti-SAT design, the CAS-Lock design uses specific complementary functions that are implemented by a cascade of AND and OR gates. Our design is more generalized in the sense that i) both complementary and non-complementary functions are allowed, and ii) the functions have a very large number of variations and they are not limited to certain type of structures. Besides the design procedures presented in Section IV, other f and g functions are possible by choosing true sets of other cardinalities and/or corresponding to alternative groups of cells in the K-map. Actually, the CAS-Lock design is a special case of our complementary design with the true set of f consisting of consecutive cells of the K-map in numerical order starting with the all-'0' cell. The CAS-Lock block can be attacked by the CAS-Unlock attack [28], which uses either all '0' or all '1' as the key. Such a key leads to the correct output for all input patterns because the right keys for the two complementary functions are the same. This attack applies to the Anti-SAT and our complementary designs as well. To address this issue, it was proposed in [29] to replace some of the XOR gates at the inputs of the function blocks randomly by XNOR gates [29]. This scheme can be also adopted for our complementary G-Anti-SAT design. Nevertheless, our non-complementary design is not subject to the CAS-Unlock attack since the keys to the two functions are not the same. Another advantage of our design is that no specific structure is required in the two functions. As a result, our design is immune to any attacks that try to utilize specifics of the functions or structures.

VII. CONCLUSION

In this article, novel G-Anti-SAT schemes have been proposed by relaxing the constraints on the wrong key sets. Our schemes allow great flexibility on the two function blocks, which can be also non-complementary. Our designs are always resilient to the SAT attack. Moreover, by choosing functions whose true sets have larger cardinalities, the output corruptibility is effectively increased. As a result, our design has better resistance to the AppSAT and other attacks without compromising the SAT-attack resiliency. Methodologies have also been provided for designing the G-Anti-SAT blocks and deciding the right keys using the K-map. Future work will monitor new attacks and extend our proposed designs.

APPENDIX A

The true sets of non-complementary f and g have a common cell. Hence, $\mathbf{F}^F \cup \mathbf{G}^F \subset \mathbf{X}_n$. Accordingly, $\mathbf{F}^T_{K_f}$ needs to be a subset of $\mathbf{G}^F_{K_g}$ in order to satisfy (8). Let $\mathbf{C}$ be a subset of $\mathbf{G}^F_{K_g}$ that equals $\mathbf{F}^T_{K_f}$. Assume that $|\mathbf{C}| = |\mathbf{F}^T_{K_f}| = m$, and for elements $C_i \in \mathbf{C}$ and $F^T_{K_f,i} \in \mathbf{F}^T_{K_f}$ $(i = 1, 2, \cdots, m)$

$$C_1 = F^T_{K_f,1}$$
$$\vdots$$
$$C_n = F^T_{K_f,m}.$$

Since $\mathbf{C} \subset \mathbf{G}^F_{K_g}$, according to the definition of $\mathbf{G}^F_{K_g}$, C_i can be written as $G^F_i \oplus K_g$ for some $G^F_i \in \mathbf{G}^F$. Similarly, $F^T_{K_f,i} = F^T_i \oplus K_f$. Then the above equations can be rewritten as

$$G^F_1 \oplus K_g = F^T_1 \oplus K_f$$
$$\vdots$$
$$G^F_m \oplus K_g = F^T_m \oplus K_f.$$

Moving K_g from the left side to the right side of the equations, it can be derived that

$$\begin{aligned} G_1^F &= F_1^T \oplus K \\ &\vdots \\ G_m^F &= F_m^T \oplus K, \end{aligned} \tag{15}$$

where $K = K_f \oplus K_g$. Adding any two equations listed in (15) leads to $G_i^F \oplus G_j^F = F_i^T \oplus F_j^T$. Let $\mathbf{S} = \{G_1^F, G_2^F, \cdots, G_m^F\}$. Apparently, $\mathbf{S} \subset \mathbf{G}^F$. Therefore,

$$\exists \mathbf{S} \subset \mathbf{G}^F,\ s.t.\ \forall S_i, S_j,\ S_i \oplus S_j = F_i^T \oplus F_j^T. \tag{16}$$

According to the definition of binary distance structure in (9), (16) can be translated to Constraint 2.

REFERENCES

[1] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proc. IEEE*, vol. 102, no. 8, pp. 1283–1295, Aug. 2014.

[2] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor, and Y. Makris, "Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain," *Proc. IEEE*, vol. 102, no. 8, pp. 1207–1228, Aug. 2014.

[3] R. P. Cocchi, J. P. Baukus, L. W. Chow, and B. J. Wang, "Circuit camouflage integration for hardware IP protection," in *Proc. 51st Annu. Design Autom. Conf. Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2014, pp. 1–5.

[4] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2013, pp. 709–720.

[5] R. Torrance and D. James, "The state-of-the-art in semiconductor reverse engineering," in *Proc. 48th Design Autom. Conf. (DAC)*, New York, NY, USA, 2011, pp. 333–338.

[6] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending piracy of integrated circuits," in *Proc. Design, Autom. Test Eur.*, Munich, Germany, Mar. 2008, pp. 1069–1074.

[7] J. Rajendran *et al.*, "Fault analysis-based logic encryption," *IEEE Trans. Comput.*, vol. 64, no. 2, pp. 410–424, Feb. 2015.

[8] M. Yasin, J. J. Rajendran, O. Sinanoglu, and R. Karri, "On improving the security of logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 9, pp. 1411–1424, Sep. 2016.

[9] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proc. 49th Annu. Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2012, pp. 83–89.

[10] J. B. Wendt and M. Potkonjak, "Hardware obfuscation using PUF-based logic," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, Nov. 2014, pp. 270–271.

[11] Y. Lee and N. A. Touba, "Improving logic obfuscation via logic cone analysis," in *Proc. Latin-Amer. Test Symp.*, Puerto Vallarta, Mexico, 2015, pp. 1–6.

[12] S. Khaleghi, K. D. Zhao, and W. Rao, "IC piracy prevention via design withholding and entanglement," in *Proc. 20th Asia South Pacific Design Autom. Conf.*, Chiba, Japan, Jan. 2015, pp. 821–826.

[13] B. Liu and B. Wang, "Embedded reconfigurable logic for ASIC design obfuscation against supply chain attacks," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Dresden, Germany, 2014, pp. 1–6.

[14] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, Washington, DC, USA, May 2015, pp. 137–143.

[15] Y. Xie and A. Srivastava, "Anti-SAT: Mitigating SAT attack on logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 2, pp. 199–207, Feb. 2019.

[16] M. Yasin, B. Mazumdar, J. J. V. Rajendran, and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2016, pp. 236–241.

[17] K. Shamsi, T. Meade, M. Li, D. Z. Pan, and Y. Jin, "On the approximation resiliency of logic locking and IC camouflaging schemes," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 2, pp. 347–359, Feb. 2019.

[18] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal attacks on logic locking and camouflaging techniques," *IEEE Trans. Emerg. Topics Comput.*, vol. 8, no. 2, pp. 517–532, Apr. 2020.

[19] Y. Shen, A. Rezaei, and H. Zhou, "SAT-based bit-flipping attack on logic encryptions," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Dresden, Germany, Mar. 2018, pp. 629–632.

[20] H. Zhou, "A humble theory and application for logic encryption," *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 696, 2017.

[21] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Dallas, TX, USA, Oct. 2017, pp. 1601–1618.

[22] Y. Shen, A. Rezaei, and H. Zhou, "A comparative investigation of approximate attacks on logic encryptions," in *Proc. 23rd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jeju, South Korea, Jan. 2018, pp. 271–276.

[23] D. Sirone and P. Subramanyan, "Functional analysis attacks on logic locking," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Florence, Italy, Mar. 2019, pp. 936–939.

[24] 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, no. 10, pp. 2778–2786, Oct. 2019.

[25] B. Shakya, X. Xu, M. Tehranipoor, and D. Forte, "CAS-lock: A security-corruptibility trade-off resilient logic locking scheme," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2020, no. 1, pp. 175–202, Nov. 2019.

[26] *Netlist Encryption and Obfuscation Suite*. Accesssed: Feb. 13, 2021. [Online]. Available: https://bitbucket.org/kavehshm/neos/src/master/

[27] X. Xu, B. Shakya, M. M. Tehranipoor, and D. Forte, "Novel bypass attack and BDD-based tradeoff analysis against all known logic locking attacks," *Cryptograph. Hardw. Embedded Syst.*, vol. 10529, pp. 189–210, 2017.

[28] A. Sengupta and O. Sinanoglu, "CAS-unlock: Unlocking CAS-lock without access to a reverse-engineered netlist," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 1443, 2019.

[29] S. Bicky, X. Xu, M. M. Tehranipoor, and D. Forte, "Defeating CAS-unlock," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 324, 2020.

Jingbo Zhou received the B.S. degree in telecommunication engineering from the Beijing University of Posts and Telecommunications, Beijing, China. He is currently pursuing the Ph.D. degree with the Electrical and Computer Engineering Department, The Ohio State University, OH, USA.

His current research interests include hardware security and cryptography.

Xinmiao Zhang (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from the University of Minnesota. Between her academic positions, she was a Senior Technologist with Western Digital/SanDisk Corporation. She was a Timothy E. and Allison L. Schroeder Assistant Professor from 2005 to 2010 and an Associate Professor from 2010 to 2013 with Case Western Reserve University. In 2017, she joined as an Associate Professor with The Ohio State University. She authored the book *VLSI Architectures for Modern Error-Correcting Codes* (CRC Press, 2015), and co-edited *Wireless Security and Cryptography: Specifications and Implementations* (CRC Press, 2007). She also published more than 100 articles. Her research interests include VLSI architecture design, digital storage and communications, security, and signal processing.

Dr. Zhang is a member of the CASCOM and VSA technical committees. She was also a Co-Chair of the Data Storage Technical Committee from 2017 to 2020. She received an NSF CAREER Award in January 2009. She was also a recipient of the Best Paper Award at 2004 ACM Great Lakes Symposium on VLSI and the 2016 International SanDisk Technology Conference. She has been an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I from 2010 to 2019 and the IEEE OPEN JOURNAL OF CIRCUITS AND SYSTEMS since 2019. She was elected to serve on the Board of Governors of the IEEE Circuits and Systems Society from 2019 to 2021. She served on the technical program and organization committees for many conferences, including ISCAS, SiPS, ICC, GLOBECOM, GlobalSIP, and GLSVLSI.