Generalized SAT-Attack-Resistant Logic Locking

Jingbo Zhou and Xinniao Zhang, Senior Member, IEEE

Abstract—Logic locking is used to protect integrated circuits (ICs) from piracy and counterfeiting. An encrypted IC implements 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-SA T 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 AppSA T and removal attacks. This paper proposes a generalized (G-)Anti-SA T scheme. Different from the Anti-SA T scheme, a variety of complementary or non-complementary functions can be adopted for the two blocks in our G-Anti-SA T scheme. Pairs of functions that consist of similar number of minterms can be chosen to resist the AppSA T and removal attacks. Meanwhile, our design requires the same number of iterations in the SAT attack as the Anti-SA T scheme, and hence is always resistant to the SAT attack. The Anti-SA T scheme is just a special case of our proposed design.

Index Terms—Anti-SA T, AppSA T attack, Hardware security, Logic locking, Removal attack, SAT attack

1 INTRODUCTION

NOWADAYS, integrated circuits (ICs) are designed and produced in a multi-vendor environment, which makes the designs face various security threats. In particular, nestlists 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 [3] by making functionally different logic gates look alike in layout. However, unlike logic encryption/locking [5], it is ineffective when the netlist is available. The basic idea of logic locking is to insert key-controlled logic gates 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], [8], 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 it is utilized to identify wrong keys. For many logic locking schemes, only a small number of DIPs are needed to identify all wrong keys. As a result, the SAT attack can be done within a few hours even if the key size is not very large.

Several schemes have been proposed to resist the SAT attack in recent years [15], [16]. The main idea is to adopt functional blocks that make the number of iterations in the SAT attack exponential. The Anti-SA T 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

1) Generalized constraints on the two blocks of functions that make the number of the SAT attack iterations exponential to the key size are derived.
2) Generalization is made in a second dimension to
allow the two functions to be non-complementary. Constraints on the existence of correct keys for this case are derived.

3) Methodologies are developed to design functions satisfying the generalized constraints using K-maps.

4) Guidelines are provided to design functions that are resistant to AppSAT and removal attacks and have reduced logic complexity.

This paper is organized as follows. Section 2 provides background of the attacks and Anti-SAT design. Section 3 proposes our G-Anti-SAT constraints. Section 4 presents methodologies for developing functions satisfying the G-Anti-SAT constraints using K-maps. Guidelines for resisting removal and AppSAT attacks and reducing logic complexity are also discussed in this section. Analysis and experimental results showing the resistance of our design to various attacks are given in Section 5. Discussions and conclusions follow in Section 6 and Section 7, respectively.

2 Background

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

2.1 SAT attack

The SAT attack [14] is a powerful technique against logic locking. The attack model assumes that the attacker has access to the locked gate-level netlist, which can be obtained by reverse engineering or from an un-trusted foundry. Define the locked netlist as \( \vec{Y} = f_o(\vec{X}, \vec{K}) \) with primary inputs \( \vec{X} \), key inputs \( \vec{K} \), and primary outputs \( \vec{Y} \). Its CNF formula is represented as \( C_o(\vec{X}, \vec{K}, \vec{Y}) \). It is also assumed that the attacker has an activated chip. Its netlist is represented as \( \vec{Y} = f_o(\vec{X}) \), and its CNF formula is \( C_o(\vec{X}, \vec{Y}) \).

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

\[
F_0 := C_o(\vec{X}, \vec{K}^c_1, \vec{Y}^c_1) \land C_o(\vec{X}, \vec{K}^c_2, \vec{Y}^c_2) \land (\vec{Y}^c_1 \neq \vec{Y}^c_2)
\]  

(1)

to solve for an \( \vec{X} \) that leads to different outputs, \( \vec{Y}^c_1 \) and \( \vec{Y}^c_2 \), under two different keys, \( \vec{K}^c_1 \) and \( \vec{K}^c_2 \). This \( \vec{X} \) is referred to as the DIP and is denoted by \( \vec{X}_d^c \). Then the logic function of the activated circuit is utilized to get the corresponding correct output \( \vec{Y}^d = f_o(\vec{X}_d^c) \). After that, new constraints corresponding to \( \vec{X}^d_j \) and \( \vec{Y}^d_j \) are added and the original SAT formula in (1) is updated as \( F_1 = F_0 \land C_o(\vec{X}^d_j, \vec{K}_1, \vec{Y}^d_j) \land C_o(\vec{X}_d^c, \vec{K}_2, \vec{Y}^d_j) \). Then the updated SAT formula is solved for DIPs and the DIPs are utilized to update the SAT formula iteratively. In the \( i \)th iteration, the SAT formula is

\[
F_i = F_0 \bigwedge_{j=1}^{i} (C_o(\vec{X}^d_j, \vec{K}_1, \vec{Y}^d_j) \land C_o(\vec{X}^d_j, \vec{K}_2, \vec{Y}^d_j)).
\]

If \( F_i \) is satisfiable, then there exist at least one pair of keys \( \vec{K}^c_1, \vec{K}^c_2 \), and \( \vec{X}^d_{i+1} \) such that \( f_o(\vec{X}^d_{i+1}, \vec{K}^c_1) \neq f_o(\vec{X}^d_{i+1}, \vec{K}^c_2) \), which means not all wrong keys have been excluded from the key space. When the SAT formula is no longer solvable in an iteration, say \( \lambda \), the algorithm stops. At this time, the correct key can be derived by solving the following SAT formula

\[
F := \bigwedge_{i=1}^{\lambda} C_o(\vec{X}^d_i, \vec{K}, \vec{Y}^d_i).
\]  

(2)

2.2 Anti-SAT block

An 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 \( \vec{X} \) 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 \( \vec{K}_1 = [k_1, k_2, \ldots, k_n] \) and \( \vec{K}_2 = [k_{n+1}, k_{n+2}, \ldots, k_{2n}] \). Any \( \vec{K}_1, \vec{K}_2 \) are correct 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 \( \vec{X} = [x_1, x_2, \ldots, x_n] \). The input to the \( g \) function in Fig. 1(a) is \( \vec{L} = \vec{X} \oplus \vec{K}_1 \). Define

\[
\mathcal{L}^T = \{ \vec{L}|g(\vec{L}) = 1 \}, \quad (|\mathcal{L}^T| = p)
\]

\[
\mathcal{L}^F = \{ \vec{L}|g(\vec{L}) = 0 \}, \quad (|\mathcal{L}^F| = 2^n - p)
\]

(3)

In the remainder of this paper, \( \mathcal{L}^T \) 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

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

(4)

When \( p = 1 \) or \( 2^n - 1 \), \( \lambda \geq 2^n \). Since there are \( 2^n \) input combinations, this means that all possible input patterns need to be gone through as DIPs to reveal the correct keys and 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.

Two methods have been proposed in [15] to integrate the Anti-SAT block into a circuit: random integration and secure integration. They use random signals in the circuit and primary inputs of the circuit, respectively, as the inputs to the Anti-SAT block. Unlike the random integration, the secure integration guarantees that the number of iterations needed by the SAT attack is maximized.

2.3 AppSAT attack

In the AppSAT attack [18], the corruptibility of the output signal is defined as

\[
Cr = \Pr_{\vec{X} \in \mathcal{X}, \vec{K} \in \mathcal{K}}[C_o(\vec{X}, \vec{K}) \neq C_o(\vec{X})],
\]  

(5)
where $\mathcal{X}$ and $\mathcal{K}$ are the sets of all possible input and key patterns, respectively. The Anti-SAT block in Fig. 1 has one output signal. This signal is connected to the circuit to be locked and the locked circuit functions correctly if this signal has the correct value. Since $g$ and $\overrightarrow{f}$ are $n$-input AND and NAND gates, respectively, for every wrong key, there is only one input pattern that makes the output of the Anti-SAT block wrong. Accordingly, the Anti-SAT design has low corruptibility.

The AppSAT attack avoids exponential number of iterations by introducing random query reinforcement and stopping the query process early. After every certain number of iterations, random input query patterns are inserted and additional constraints from the random 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.

2.4 Removal attack

The removal attack [17] can be utilized to identify the last gate, $G$, in the Anti-SAT block. Then the output of this gate is replaced by the correct signal, which is ‘0’ and ‘1’ for the type-0 and type-1 blocks, respectively, in the circuit adopting the Anti-SAT block.

The removal attack is carried out using signal probability skew (SPS). The SPS value of a signal, $x$, is defined as

$$s_x = Pr[x = 1] - 0.5.$$ (6)

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

$$ADS = |s_1 - s_2|.$$ (7)

Assuming that $\overrightarrow{X}, \overrightarrow{K}_1, \overrightarrow{K}_2$ are random. Then the SPS values of the inputs to the XOR gates in the Anti-SAT block are zero. According to (6), 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 $s_{\negAND} = \prod_{i=1}^{n}(0.5 + s_i) - 0.5$, where $s_i$ is the SPS of the $i^{th}$ input. Since $s_i = 0$ for the AND gate in the function, $s_{\negAND}(\overrightarrow{X}, \overrightarrow{K}_1) = 0.5^n - 0.5$. As $n \to \infty$, $s_{\negAND}(\overrightarrow{X}, \overrightarrow{K}_1) \approx -0.5$. Similarly, for the $n$-input NAND gate output from $\overrightarrow{y}$, the SPS is $s_{\negNAND}(\overrightarrow{X}, \overrightarrow{K}_2) = 0.5 - 0.5^n$. It 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 value is $-0.5$ and ADS value of final gate $G$ is $ADS_G = |s_{\negAND}(\overrightarrow{X}, \overrightarrow{K}_1) - s_{\negNAND}(\overrightarrow{X}, \overrightarrow{K}_2)| \approx 1$, if the number of inputs to the Anti-SAT block is large.

It was found in [17] 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. To remove the $G$ gate, its output signal is replaced by 0 or 1 in the circuit when the SPS of $G$ is negative or positive, respectively.

It was also mentioned in [17] that the removal and AppSAT attack can be combined. The combined attack can return an exact unlocked netlist, instead of an approximate key when using the AppSAT attack alone.

3 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 true set as defined in (3), is either too small or too big. 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, which enables our design to be effectively resistant to the AppSAT and removal attacks at the same time.

3.1 Wrong key sets analysis

Define $\mathcal{WK}_{\overrightarrow{X}}$ as the wrong key set that input $\overrightarrow{X}$ can exclude. In other words, $\mathcal{WK}_{\overrightarrow{X}} = \{\overrightarrow{K} \mid f_{\overrightarrow{X}}(\overrightarrow{K}, \overrightarrow{X}) \neq f_{\overrightarrow{X}}(\overrightarrow{X_i})\}$. If there exist $\overrightarrow{X}_i, \overrightarrow{X}_j$ such that $\mathcal{WK}_{\overrightarrow{X_i}} = \mathcal{WK}_{\overrightarrow{X_j}}$, and the algorithm has already selected one of them as a DIP, then the other one will not be 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 inputs that have the same wrong key sets and are not selected as DIPs.

Consider a circuit that has $n$ inputs and requires $\lambda$ iterations in the SAT attack. Denote the set of DIPs by $\mathcal{X}_{DIP}$. Accordingly, $\lambda = |\mathcal{X}_{DIP}|$. If a block needs to be resistant to the SAT attack, $\lambda$ needs to be as big as possible, which is $2^n$. This means that each possible $n$-bit input combination 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 (3) is 1, for each possible input $\overrightarrow{X}_i$, $|\mathcal{WK}_{\overrightarrow{X_i}}| = 15$. The key input has 8 bits and hence $2^8$ different combinations. 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 $\lambda \geq \frac{240}{16} = 16$. On the other hand, for 4-bit input, there are $2^4$ combinations. Hence, $\lambda = 16$. Apparently, for this Anti-SAT scheme, the wrong key sets for different input combinations do not have any overlap. In other words,

$$\forall \overrightarrow{X}_i \neq \overrightarrow{X}_j \in \mathcal{X}_n, \mathcal{WK}_{\overrightarrow{X}_i} \cap \mathcal{WK}_{\overrightarrow{X}_j} = \emptyset,$$ (8)

where $\mathcal{X}_n$ is the set of all possible inputs of $n$ bits.

$\lambda$ may 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.

3.2 Highlights of proposed generalized Anti-SAT block

The proposed G-Anti-SAT block generalizes the previous approach by allowing the wrong key sets of different input
combinations to have overlaps. In general, instead of (8), our design requires that
\[ \forall X_i \neq X_j \in X_n, \exists \bar{L} \in \mathbb{W} \mathbb{K}_X \& \bar{L} \notin \mathbb{W} \mathbb{K}_X. \] (9)

In other words, each wrong key set has at least one distinct element. Adopting the above relaxation, there are still \( \lambda = 2^n \) DIPs. Hence, our generalized design is still resistant to the SAT attack. In order to allow overlapping wrong key sets, the cardinality of the true set \( |L^T| \) is relaxed so that it can be integers other than 1 or \( 2^n - 1 \). A second dimension of generalization is done by allowing the two functions to be \( f \) and \( g \), which are not necessarily complementary of each other. By choosing true sets with medium cardinality, our design leads to output with high corruptibility and hence is resistant to the AppSAT attack. Also the ADS value of the final gate in our scheme can be tuned by changing the cardinalities of the true sets of \( f \) and \( g \). By using two functions whose true set cardinalities are as close to each other as possible, our design is immune to the removal attack, which is not addressable by the previous Anti-SAT block. Similarly, our proposed design can be also integrated into circuits using the methods in [15].

In the following, subsection 3.3.1 analyzes the constraints on the true sets to satisfy (9). 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 3.3.2. Section 4 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 identify the truth sets leading to lower hardware implementation complexity for the proposed G-Anti-SAT block.

3.3 Constraints for SAT attack resistance and correct key existence

3.3.1 Constraints for resisting SAT attacks

Fig. 2 shows our proposed G-Anti-SAT block for type-0 design. Different from the previous design, 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. 2. All the proposed analysis and constraints can be extended easily for the type-1 design.

Define \( L^f = \{ \bar{L} | f(L) = 1 \} \), \( L^g = \{ \bar{L} | g(L) = 1 \} \), \( L^T = \{ \bar{L} | f(L) = 0 \} \) and \( L^T = \{ \bar{L} | g(L) = 0 \} \).

Following the convention in [15], ‘1’ is considered as the incorrect output for a type-0 block. For the architecture in Fig. 2, the output function is \( y = f(X \oplus \bar{K}_f) \land g(X \oplus \bar{K}_g) \) where \( \bar{K}_f \) and \( \bar{K}_g \) are the key inputs of block \( f \) and \( g \), respectively. \( y = 1 \) only if \( L_f^T \subseteq \bar{X} \oplus \bar{K}_f \in L^fT \) and \( L_g^T \subseteq \bar{X} \oplus \bar{K}_g \in L^gT \). Therefore, the wrong key combinations in \( \mathbb{W} \mathbb{K}_X \) are in the format of \( [\bar{X} \oplus L_f^T \mid \bar{X} \oplus L_g^T] \), where \( | \) means concatenation. Accordingly, (9) can be interpreted as

\[ \forall X_i \neq X_j \in X_n, \exists \bar{L}_i^f \neq \bar{L}_j^f \in L^fT, \bar{L}_i^g \neq \bar{L}_j^g \in L^gT \] s.t. \( [\bar{L}_i^f \oplus \bar{X}_i \mid \bar{L}_i^g \oplus \bar{X}_i] \neq [\bar{L}_j^f \oplus \bar{X}_j \mid \bar{L}_j^g \oplus \bar{X}_j]. \) (11)

Since \( X_n \) includes all \( n \)-bit vectors, for any \( \bar{L}_i^f \neq \bar{L}_j^f \), there must exist \( \bar{X} \in X_n \) such that \( \bar{L}_i^f \oplus \bar{X} = \bar{X} \). \( \bar{X} \) can be also rewritten as the sum of two elements in \( X_n \), i.e., \( \bar{X} = \bar{X}_i \oplus \bar{X}_j \). The constraint \( \forall X_i \neq X_j \in X_n \), \( L_f^T \oplus \bar{X}_i \neq L_f^T \oplus \bar{X}_j \) can never be satisfied. Similarly, \( L_g^T \oplus \bar{X}_i \neq L_g^T \oplus \bar{X}_j \) for \( \forall X_i \neq X_j \) is not true. Therefore, to satisfy the constraints in (11), \( L_f^T \) and \( L_g^T \) need to be designed jointly so that \( L_f^T \oplus \bar{X}_i = L_f^T \oplus \bar{X}_j \) and \( L_g^T \oplus \bar{X}_i = L_g^T \oplus \bar{X}_j \) are not true at the same time.

Define the binary distance between two vectors \( \bar{X}_1 \) and \( \bar{X}_2 \) as \( \bar{d} = \bar{X}_1 \oplus \bar{X}_2 \). Let \( D_{\bar{e} \rightarrow S} \) be a set consisting of binary distances between an element \( \bar{e} \) and all the other elements in \( S \). In other words,

\[ D_{\bar{e} \rightarrow S} = \{ \bar{d} = \bar{e} \oplus \bar{e}_i | \bar{e}_i \neq \bar{e}, \bar{e}_i \in S \}. \] (12)

Then \( D_{L_f^f \rightarrow L_f^T} \) is the set of vectors consisting of \( \bar{L}_i^f \oplus \bar{L}_j^f \) for every \( L_f^f \in L^fT \) and \( L_f^f \neq L_j^f \). If \( \bar{L}_i^f \oplus \bar{X}_i = L_f^f \oplus \bar{X}_i \) is satisfied, \( \bar{L}_j^f \oplus \bar{L}_j^f = \bar{X}_i \oplus \bar{X}_j \). Hence, \( \bar{X}_i \oplus \bar{X}_j \) is also in the set \( D_{L_f^f \rightarrow L_f^T} \). Similarly, the \( \bar{L}_i^g \oplus \bar{X}_i \) of the \( \bar{X}_i \) and \( \bar{X}_j \) satisfying the constraint that \( \bar{L}_i^g \oplus \bar{X}_i = \bar{L}_j^g \oplus \bar{X}_j \) is in the set \( D_{L_g^g \rightarrow L_g^T} \). Accordingly, the constraints in (11) are translated to

\[ \text{Constraint 1: } D_{L_f^f \rightarrow L_f^T} \cap D_{L_g^g \rightarrow L_g^T} = \emptyset. \] (13)

The above equation gives constraints equivalent to (9). On the other hand, these constraints can be utilized to construct \( L^fT \) and \( L^gT \) more easily. They are referred to as Constraint 1 in the reminder of this paper.

From Constraint 1, it is clear that the functions \( f \) and \( g \) do not have to be complementary, and \( |L^fT|, |L^gT| \) 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 \( L^fT \) first. For the selected \( L^fT \), the choice of \( L^gT \) may not be unique. Any \( L^gT \) satisfying Constraint 1 can be utilized to be resistant to the SAT attack.

Example 1 Take the structure in Fig. 2 with 4-bit input as an example. Different from the previous design, the \( f \) and \( g \) functions are allowed to be non-complementary. First,
It was found that to make $L_{K_f} \supseteq L_{f_T}$, $L_g$ should have a subset with the same binary distance structure as $L_f$. In other words,

$$\text{Constraint 2: } \exists S \subset L_g, \text{ s.t. } D_S = D_{L_f}$$ (16)

The proof is detailed in the appendix.

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

1) In Example 1, $L_{f_T} = [0, 1, 2, 3]$ and $L_g = [0, 8, 9, 11, 10]$. From (15), it can be computed that $D_{L_f} = [1, 2, 3, 3, 2, 1, 1, 2, 3, 3, 2, 1]$. The subset $S$ of $L_g$ that have the same binary structure with $L_f$ can be $[12, 13, 14, 15]$. $D_S = [1, 2, 3, 3, 2, 1, 1, 2, 3, 3, 2, 1]$. Hence, this block has right keys. The method to find the right keys will be presented in Section 4. It can be found that one of the right keys is $K_f = [0, 0, 0, 0]$ and $K_g = [0, 0, 0, 1]$.

2) In Example 3, $f(\bar{L}) = \{0, 1\}$ and $g(\bar{L}) = \{1, \bar{1}\}$. It can be calculated that $D_{L_f} = [1, 2, 3, 3, 2, 1, 1, 2, 3, 3, 2, 1]$. However, there is no subset of $D_{L_g}$ having the same binary structure as $L_f$. Hence correct key does not exist.

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

$$\exists L_{f_T}, L_g \text{ s.t. } D_{L_{f_T}} - L_{f_T} \cap D_{L_g} = \emptyset, \exists S \subset L_g \text{ s.t. } D_S = D_{L_{f_T}}$$

4 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 correct 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.

4.1 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. 2 with 4-bit input $\bar{L} = \{l_3, l_2, l_1, l_0\}$ to $f$ and $g$ as an example. The corresponding K-map has 16 cells represented as a $4 \times 4$ array. $l_3 l_2 l_1 l_0$ are used to label the columns and rows, respectively, as shown in Fig. 3.

Constraint 1 requires that there exist $L_{f_T}$ and $L_g$ satisfying $D_{L_{f_T}} \cap D_{L_g} = \emptyset$. When $L_{f_T}$ and $L_g$ are non-complementary, the groups of cells for $L_{f_T}$ and $L_g$ can have overlaps in the K-map and a common cell can be used as both $L_{f_T}$ and $L_g$. In a K-map, the column and row labels for each cell are distinct. Hence, adding the
It is shifted and/or flipped according to the $L$ equivalent to (14). In the K-map, the group of cells for $\vec{K}$ and/or flipped according to that the shifted and/or flipped groups for $L$ do not need to cover all the cells.

When $f$ and $g$ are non-complementary, additional constraints need to be added to K-map cell selection in order to satisfy Constraint 2 and hence have correct keys. Without loss of generality, consider that $L^f$ consists of at most half of the cells in the K-map. For the common cell shown in Fig. 3(a), randomly select cells in the left half of the K-map to be $L^f$, such as those shown in Fig. 3(a). Two symmetric groups of cells in the K-map always have the same binary distance structure defined in (12). One example of the group of cells that is symmetric to those for the $L^f$ in Fig. 3(a) is shown in Fig. 3(b). Having $L^g$ include at least the symmetric cells in Fig. 3(b) would satisfy Constraint 2. On the other hand, $L^g$ covers all the other cells not covered by $L^f$ and can only share one common cell with $L^f$. Therefore, $L^g$ also needs to cover every cell in $L^f$ except the common cell. For the $L^f$ selected in Fig. 3(a), Fig. 3(b) shows the $L^g$ satisfying these requirements. The rest cells, as shown in Fig. 3(d), form $L^g$ satisfying Constraint 1 and 2. It should be noted that there may exist other choices of $L^f$ and $L^g$ satisfying Constraint 1 and 2 besides the ones can be located by using the above method.

The correct keys $\vec{K}_f$ and $\vec{K}_g$ can be easily decided from the cells for $L^f$ and $L^g$ in the K-map. Constraint 2 is equivalent to (13). In the K-map, the group of cells for $L^f$ has the same shape as that for $L^g$, except that it is shifted and/or flipped according to the $\vec{K}_f$ vector. Similarly, the group of cells for $L^g$ is that for $L^g$ shifted and/or flipped according to $\vec{K}_g$. Then (13) is translated to that the shifted and/or flipped groups for $L^f$ and $L^g$ need to cover every cell in the K-map. The vectors leading to such shifting/flipping are the correct keys $\vec{K}_f$ and $\vec{K}_g$.

For the $L^f$ in Fig. 3(a), Fig. 4(a) shows the cells for the corresponding $L^f$. The cells for $L^g$ are illustrated in Fig. 4(b). It can be seen that the union of such $L^g$ and $L^g$ covers every cell in the K-map except the one with $[l_3, l_2, l_1, l_0] = [0, 1, 1, 1]$, which is the common cell. One way to cover each cell in the K-map is to keep the cells for $L^g$ unchanged, which means $\vec{K}_g = [0, 0, 0, 0]$, and use $\vec{K}_f = [1, 0, 0, 0]$, which leads to the group of cells of $L^f$ shown in Fig. 4(b). The gray cells in Fig. 4(c) and Fig. 4(b) are $L^g_{K_g}$ with $\vec{K}_g = [0, 0, 0, 0]$ and $L^f_{K_f}$ with $\vec{K}_f = [1, 0, 0, 0]$, respectively. They cover all cells in the K-map. There are many choices of $\vec{K}_f$ and $\vec{K}_g$ that satisfy (13). Another example is that $\vec{K}_g = [1, 0, 0, 0]$ and $\vec{K}_f = [0, 0, 0, 0]$. It corresponds to that the $L^f$ in Fig. 4(a) is unchanged and the $L^g$ in Fig. 4(c) is flipped horizontally in the K-map.
4.2 K-map cell selection for complementary functions

When \( f \) and \( g \) are complementary, \( L_f^T \) and \( L_g^T \) 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. 5(a). This column is referred to as the dividing column in this paper. The labels for each column in the K-map are distinct. Hence, adding the column labeling 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 \( L_f^T \), \( L_g^T \) and picking \( L_f^{T_1} \) and \( L_g^{T_1} \) from the dividing column would satisfy Constraint 1. To reduce the logic complexity of \( f \) and \( g \), adjacent columns should be put into \( L_f^T \) and \( L_g^T \), and the numbers of columns in \( L_f^T \) and \( L_g^T \) should be as close as possible. For example, in Fig. 5(b), the first column is put in \( L_f^T \) and the third and fourth columns are put in \( L_g^T \).

Next, the cells in the dividing column should be split between \( L_f^T \) and \( L_g^T \). 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 \( L_g^T \) and the other cells are put in \( L_f^T \), as shown by the example in Fig. 5(c). The cell of \( L_g^T \) in the dividing column can be used as \( L_{g_1}^T \) and any other cells in the dividing column can be used as \( L_{f_1}^T \). The sum of the column labels of any two cells in the same column is zero. Hence, adding \( L_{f_1}^T \) to any other cells in the dividing column would result in a zero column label, and any vector in \( \mathbf{D}_{L_{f_1}^T-L_g^T} \) is different from those in \( \mathbf{D}_{L_f^T-L_g^T} \). As a result, \( L_f^T \) and \( L_g^T \) from such K-map cells splitting satisfy Constraint 1.

When \( f \) and \( g \) are complementary, any \( \mathbf{K}_f = \mathbf{K}_g \) can be used as a correct key. Constraint 2 does not need to be considered in this case.

4.3 True sets design for complexity reduction and attack resistance

To reduce the implementation complexity, adjacent cells should be put into the two true sets \( L_f^T \) and \( L_g^T \). The larger the number of adjacent cell groups and the smaller the number of groups in the K-map that can be used to cover \( L_f^T \) and \( L_g^T \), the simpler the logic functions \( f \) and \( g \).

To increase the corruptibility for App-SAT attack resistance as defined in [5] for a type-0 G-Anti-SAT block, \( |L_f^T| \) and \( |L_g^T| \) should not be either very small, such as 1, or very large, such as \( 2^n - 1 \). On the other hand, to make the ADS value defined in [7] for the last gate in Fig. 2 close to ‘0’, which is needed to resist the removal attack, \( L_f^T \) and \( L_g^T \) should have similar cardinality. These guidelines can be followed to design the \( f \) and \( g \) blocks. In addition, the ADS of the last gate can be tuned to any value in our design by changing the relative cardinalities of \( L_f^T \) and \( L_g^T \).

5 EXPERIMENTS AND RESULTS

This section evaluates the achievable security level of the proposed G-Anti-SAT scheme. Experiments and analyses are carried out to show that the proposed design is resistant to the SAT, AppSAT, and removal attacks. The SAT attack tool in [14] based on Lingeling SAT solver is used in our experiments. The CPU time is limited to 10 hours as in [14], and the experiments are run over an Intel Core i7 with 4GB RAM. Analysis on the query process of the AppSAT is carried out to show that it is ineffective on our design. The evaluation for the removal attack resistance is done by calculating the ADS values as in [17].

5.1 G-Anti-SAT block example

To compare with the Anti-SAT scheme [15], inputs with 16 bits are adopted in our G-Anti-SAT block. Two type-0 block examples are evaluated in our experiments.

**G-Anti-SAT Block m1**: This example has non-complementary \( f \) and \( g \). For 16-bit input, the K-map has \( 2^8 \) columns and \( 2^8 \) rows. The cell in the 64th column and 170th row is randomly selected to be the common cell. To minimize the logic complexity and the ADS value for the last gate, all the cells in the first 64 columns are chosen to form \( L_f^T \). The last 64 columns have the same binary distance structure as \( L_f^T \). Hence \( L_g^T \) should include at least the first and last 64 columns, except the common cell, in order to satisfy Constraint 2. To make the ADS value of the last gate as close to 0 as possible, \( L_g^T \) should have about the same number of cells as \( L_f^T \). Hence, form \( L_g^T \) by all the cells in the first 64 columns and last 128 columns except the common cell. Accordingly, \( L_g^T \) consists of the second 64 columns in the K-map and the common cell. Denote the inputs to the \( f \) and \( g \) blocks by \([l_1, l_2, \ldots, l_{16}]\). It can be derived that \( f(\mathbf{L}) = l_{15} \land l_{14}, \quad g(\mathbf{L}) = (l_{15} \land l_{14}) \lor (l_{15} \land l_{13} \land l_{12} \land l_{11} \land l_{10} \land l_9 \land l_8 \land l_7 \land l_6 \land l_5 \land l_4 \land l_3 \land l_2 \land l_1 \land l_0) \). \( L_f^T \) includes all cells in the K-map except the first 64 columns and \( L_g^T \) contains all cells except the second 64 columns and the common cell. In order to make \( L_f^{K_f} \) and \( L_g^{K_g} \) cover every cells in the K-map, one method is to keep the cells of \( L_g^T \) unchanged, which means \( K_g = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] \), and use \( K_f = [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] \) so that the cells in \( L_g^{K_g} \) cover the forth, first, and second groups of 64 columns.

**G-Anti-SAT Block m2**: Complementary \( f \) and \( g \) functions are considered in this example. First, pick the 128th column in the K-map as the dividing column. To reduce the logic complexity, put the first 127 columns into \( L_f^T \) and the 128 columns on the right side into \( L_g^T \). One cell in the dividing column should be put into one true set and the rest cells should be put into the other set. Without loss of generality, the last cell in this column is separated from the others. To make the cardinalities of \( L_f^T \) and \( L_g^T \) as close as possible in order to resist the removal attack, the last cell of the dividing column is put into \( L_g^T \) and the other cells in this column are included in \( L_f^T \). Accordingly, \( g(\mathbf{L}) = l_{15} \lor (l_{14} \land l_{13} \land l_{12} \land l_{11} \land l_{10} \land l_9 \land l_8 \land l_7 \land l_6 \land l_5 \land l_4 \land l_3 \land l_2 \land l_1 \land l_0) \). In this case, the Anti-SAT block is also simulated
TABLE 1
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          | 255   | 4095   |        |
| time (second)       | 0.44  | 66.42  | timeout|
| complementary       |       |        |        |
| G-Anti-SAT          | 255   | 4095   |        |
| time (second)       | 0.80  | 166.42 | timeout|
| Anti-SAT block      |       |        |        |
| # of iterations     | 235   | 4095   |        |
| time (second)       | 0.82  | 175.74 | -      |

Fig. 6. The logic diagram of block \( m_1 \)

in the same hardware environment and the results are included in Table 1. It can be seen that our design achieves the same resistance to the SAT attack in terms of the number of iterations. The reason that the time consumed by the attack on the non-complementary G-Anti-SAT block is less than those of 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.

5.3 Removal attack resistance analysis

Fig. 6 shows the logic diagram of block \( m_1 \). Assume that each input is equal to ‘1’ with probability 0.5. Using (6) and (7), the ADS value for each gate in Fig. 6 can be computed. The ADS value for the last gate, \( G \), is 0.00001. Hence, our design is resistant to the removal attack. The five gates with the largest ADS values in block \( m_1 \) are listed in Table 2 and they are around 0.5. When the Anti-SAT block is incorporated into circuits, there are many gates with ADS values in a wide range and many of them are larger than 0.5. Hence, the gates in Table 2 can not be identified by examining the ADS values either.

The logic diagram of block \( m_2 \) is shown in Fig. 7. The largest ADS values of all gates in Fig. 7 are listed in Table 3. Similar to block \( m_1 \), the ADS value of the last gate is around 0 and the largest five ADS values are also around 0.5 in block \( m_2 \). As a result, block \( m_2 \) is also resistant to the removal attack.

Unlike the Anti-SAT design in [15], our proposed G-Anti-SAT block does not rely on additional measures, such as the withholding and entanglement obfuscation techniques [12] to resist the removal attack.

5.4 AppSAT attack resistance analysis

The reason that the AppSAT attack can effectively decrypt the Anti-SAT block is that for any randomly selected wrong key, there is only one input that can make the block output wrong. The corruptibility as defined in (5) is very low.

For block \( m_1 \) with 16-bit input, \(|L_f^1| = 2^{14}\) and \(|L_g^1| = 2^{14} + 1\). Since the total number of keys is \(2^{12}\), testing every key can not be finished in practical time. Hence 500 keys are randomly selected in our test. Among these keys, there are 127 keys each of which has 16384 input patterns leading to the wrong output. For block \( m_2 \), \(|L_f^2| = 2^{15} - 1\) and \(|L_g^2| = 2^{15} + 1\). Among the 500 random keys, there are 253 keys each of which has 32767 input patterns leading to the wrong output.

In order for the AppSAT to be successful, the portion of input queries leading to the wrong output needs to fall under a very small threshold in the order of \(10^{-3}\). In block \( m_1 \) and \( m_2 \), for a large portion of keys, the probability that an input query leads to the wrong output is much higher. As a result, our proposed designs are resistant to the AppSAT attack.

6 DISCUSSIONS

Our proposed G-Anti-SAT schemes enjoy great flexibility on the allowing \( f \) and \( g \) functions. In the two examples given in the previous section, the two functions are chosen so that their true set cardinalities are about the same in order to make the ADS value of the last gate close to zero, which is needed to resist the removal attack according to [17]. By
changing the relative cardinalities of the two true sets, the ADS can be also tuned to other values.

For a given number of input bits, the larger the cardinalities of the two true sets, the higher corruptibility the output of the G-Anti-SAT block. In the case that the two functions are non-complementary, in order to guarantee the existence of right keys and lower complicated functions, the cardinalities of the two true sets can be at most around $2^{n-2}$ for $n$-bit input G-Anti-SAT blocks. The block $m_1$ in the previous section is an example for this case. When the two functions are complementary, their cardinalities can be made equal to around $2^{n-1}$, as shown by block $m_2$. Although the G-Anti-SAT designs with non-complementary functions have smaller true set cardinalities compared to those with complementary functions, both of them have a high percentage of inputs leading to high probability of getting the wrong output, which makes the AppSAT attack unsuccessful.

There is another attack scheme called the bypass attack [20]. The main idea of this attack is to construct a bypass circuit that inverts the wrong output and nullifies the effect of the wrong key. The complexity of the bypass circuit increases linearly with the number of DIPs that can exclude a wrong key. Our proposed G-Anti-SAT block can be inserted into existing circuits using the same two integration methods as for the Anti-SAT design. It was pointed out in [15] that the secure integration is better at resisting the SAT attack. However, in the Anti-SAT design, each key can be ruled out by one and exactly one DIP in the secure integration. This makes the Anti-SAT design vulnerable to the bypass attack. On the contrary, for a large portion of wrong keys, there are multiple DIPs that can exclude them in our G-Anti-SAT design. As a result, unlike the Anti-SAT block, our design is also resistant to the bypass attack.

7 CONCLUSIONS

In this paper, novel G-Anti-SAT schemes have been proposed by relaxing the constraints on the wrong key sets. Compared to prior designs, our schemes allow not only great flexibility on the two function blocks, but also non-complementary functions. As a result, proper functions can be adopted to resist various attacks. In addition, methodologies using K-maps have been proposed to design the functions and finding the right keys. Experiments and analyses showed that the proposed designs with true sets of moderate and about equal cardinalities are immune to all existing attacks. Future work will monitor new attacks and extend our proposed designs.

APPENDIX A

In the case of $L f^F \cup L g^F \subset X_n, L f^F_{K_i}$ needs to be a subset of $L g^F_{K_i}$ in order to satisfy (14). Let $C$ be a subset of $L g^F_{K_i}$ that equals $L f^F_{K_i}$. Assume $|C| = |L f^F_{K_i}| = m$, and for elements $C_i \in C$ and $L f^F_{K_i}$, $L f^F_{K_i}$ is equal to $L f^F_{K_i}$, for $i = 1, 2, \cdots, m$

$$L f^F_{K_i} = L f^F_{K_i}$$

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

$$L f^F_{K_i} = L f^F_{K_i}$$

where $K = K_f \oplus K_g$. Adding any two equations listed in (17) leads to $L f^F_{K_i} \oplus L f^F_{K_i} = L f^F_{K_i} \oplus L f^F_{K_i}$. Let $S = \{L f^F_{K_1}, L f^F_{K_2}, \cdots, L f^F_{K_m}\}$. Apparently, $S \subset L f^F$. Therefore,

$$\exists S \subset L f^F, s.t. \forall S_1, S_2, S_1 \oplus S_2 = L f^F_{K_i} \oplus L f^F_{K_i}$$

According to the definition of binary distance structure in [15], [15] 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. of the IEEE, vol. 102, no. 8, pp. 1283-1295, Aug. 2014.
[2] U. Guin, et. al., “Counterfeit integrated circuits: a rising threat in the global semiconductor supply chain,” Proc. of the IEEE, vol. 102, no. 8, pp. 1207-1228, Aug. 2014.
[3] R. P. Cocchi, et. al., “Circuit camouflage integration for hardware IP protection,” Proc. of ACM/EDAC/IEEE Design Automation Conf., pp. 1-5, San Francisco, CA, U.S.A., 2014.
[4] J. Rajendran, et. al., “Security analysis of integrated circuit camouflaging,” Proc. ACM SIGSAC Conf. on Computer & Commun. Security, pp. 709-720, New York, U.S.A, 2013.
[5] R. Torraine and D. James, “The state-of-the-art in semiconductor reverse engineering,” Proc. of ACM/EDAC/IEEE Design Automation Conf., pp. 333-338, New York, NY, U.S.A, 2011.
[6] J. A. Roy, F. Koushanfar, and I. L. Markov, “EPIC: Ending piracy with withholding and entanglement,” Proc. of the IEEE, vol. 102, no. 8, pp. 1283-1295, Aug. 2014.
[7] J. Rajendran et. al., “Fault analysis-based logic encryption,” IEEE Trans. on Computers, vol. 64, no. 2, pp. 410-424, Feb. 2015.
[8] M. Yasin, et. al., “Security analysis of integrated circuit camouflage integration for hardware IP protection,” Proc. of ACM/EDAC/IEEE Design Automation Conf., pp. 1-5, San Francisco, CA, U.S.A., 2014.
[9] J. Rajendran, et. al., “Security analysis of logic obfuscation,” Proc. of ACM/EDAC/IEEE Design Automation Conf., pp. 83-89, San Francisco, CA, U.S.A., 2012.
[10] J. B. Wendt and M. Potkonjak, “Hardware obfuscation using PUF-based logic,” IEEE/ACM Intl. Conf. on Computer-Aided Design, pp. 270-271, San Jose, CA, U.S.A., 2014.
[11] Y. Lee and N. A. Touba, “Improving logic obfuscation via logic cone analysis,” Latin-American Test Symposium, pp. 1-6, Puerto Vallarta, Mexico, 2015.
[12] S. Khaleghi, K. Zhao and W. Rao, “IC piracy prevention via design withholding and entanglement,” Asia and South Pacific Design Auto. Conf., pp. 821-826, Chiba, Japan, 2015.
[13] B. Liu and B. Wang, “Embedded reconfigurable logic for ASIC design obfuscation against supply chain attacks,” Proc. Conf. on Design, Automation and Test in Europe, pp. 1-6, Dresden, Germany, 2014.
[14] P. Subramanyan, S. Ray, and S. Malik, “Evaluating the security of logic encryption algorithms,” Proc. IEEE Intl. Symp. on Hardware Oriented Security and Trust, pp. 137-143, Washington DC, U.S.A., 2015.

[15] Y. Xie and A. Srivastava, “Anti-SAT: mitigating SAT attack on logic locking,” IEEE Trans. on Computer-Aided Design of Integrated Circuits and Syst., vol. 38, no. 2, pp. 199-207, Feb. 2019.

[16] M. Yasin, et. al., “Sarlock: SAT attack resistant logic locking,” Proc. IEEE Intl. Symp. on Hardware Oriented Security and Trust, pp. 236-241, McLean, VA, U.S.A., 2016.

[17] M. Yasin, et. al., “Removal attacks on logic locking and camouflage techniques,” IEEE Trans. on Emerging Topics in Computing, pp. 1-1.

[18] K. Shamsi, et. al., “AppSAT: approximately deobfuscating integrated circuits,” IEEE Intl. Symp. on Hardware Oriented Security and Trust, pp. 95-100, McLean, VA, U.S.A., 2017.

[19] M. Tehranipoor and F. Koushanfar, “A survey of hardware trojan taxonomy and detection,” IEEE Design & Test of Computers, vol. 27, no. 1, pp. 10-25, Jan.-Feb. 2010.

[20] X. Xu, et. al., “Novel bypass attack and BDD-based tradeoff analysis against all known logic locking attacks,” IACR Cryptology ePrint Archive, 2017.

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

His current research interest is hardware security and cryptography.

Xinxiao Zhang received her Ph.D. degree in Electrical Engineering from the University of Minnesota. She joined The Ohio State University as an Associate Professor in 2017. Prior to that, she was a Timothy E. and Allison L. Schroeder Assistant Professor 2005-2010 and Associate Professor 2010-2013 at Case Western Reserve University. Between her academic positions, she was a Senior Technologist at Western Digital/SanDisk Corporation. Dr. Zhang’s research spans the areas of VLSI architecture design, digital storage and communications, security, and signal processing.

Dr. Zhang received an NSF CAREER Award in January 2009. She is also the recipient of the Best Paper Award at 2004 ACM Great Lakes Symposium on VLSI and 2016 International SanDisk Technology Conference. 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 was elected to serve on the Board of Governors of the IEEE Circuits and Systems Society for the 2019-2021 term. She is a Co-Chair of the Data Storage Technical Committee (2017-2020), and a member of the CASCOM and VSA technical committees and DISPS technical committee advisory board of IEEE. She served on the technical program and organization committees of many conferences, including ISCAS, SiPS, ICC, GLOBECOM, GlobalSIP, and GLSVLSI.

She has been an associate editor for the IEEE Transactions on Circuits and Systems-I since 2010.