



Boosting Cryptographic ICs' Side-Channel Resistance: A Formal Framework for Automatic Identification and Protection of Leaky Paths

QIZHI ZHANG, Tianjin University, Tianjin, China

YA GAO, Tianjin University, Tianjin, China

HAOCHENG MA, Tianjin University, Tianjin, China

JIAJI HE, Tianjin University, Tianjin, China

YIQUANG ZHAO, Tianjin University, Tianjin, China

XIAOLONG GUO, Kansas State University, Manhattan, United States

Side-channel analysis (SCA) attacks pose a significant threat to cryptographic integrated circuits (ICs). While designers have endeavored to introduce various countermeasures during the IC development phase, many of these solutions incur substantial overheads in terms of area, power, and performance. Additionally, they often necessitate a full-custom circuit design for effective deployment. This issue arises due to the absence of systematic methodologies and analytical tools for circuit designers to accurately identify the sources of side-channel leakage within the hardware design. In this article, we propose the concept of side-channel tracking logic and, building upon this foundation, introduce a novel framework that seamlessly integrates with commercial design flows to automatically identify and safeguard leaky paths. Our approach begins by pinpointing partial logic cells that exhibit the highest information leakage using dynamic correlation analysis. Subsequently, formal-based leakage property checking constructs comprehensive leaky paths centered on these cells. In this process, side-channel tracking logic was proposed and applied for the first time to trace and extract side-channel leakage paths. Based on this, an automated formal modeling and leakage property verification tool was designed. Once these paths are discerned, we deploy apt hardware countermeasures, encompassing Boolean masking and random precharge, to eradicate information leakage along these routes. This framework has been experimentally validated across different encryption circuits and the efficacy of our methodology is corroborated through both simulated and real-world measurements on FPGA implementations. Empirical results showcase an enhancement of over 1000 \times in side-channel resistance, incurring a modest overhead of less than 6.53 % across power, area, and performance metrics.

CCS Concepts: • Hardware → Design modules and hierarchy; • Security and privacy → Side-channel analysis and countermeasures; Formal security models;

Additional Key Words and Phrases: CAD for security, side channel analysis, countermeasure, formal-based property checking

This research was funded in part by the National Key R&D Program of China under Grant (2023YFB4403000) and the National Natural Science Foundation of Tianjin City (22JCQNJC00970).

Authors' Contact Information: Qizhi Zhang, Tianjin University, Tianjin, Tianjin, China; e-mail: qizhi_zhang@tju.edu.cn; Ya Gao, Tianjin University, Tianjin, Tianjin, China; e-mail: gaoyaya@tju.edu.cn; Haocheng Ma, Tianjin University, Tianjin, Tianjin, China; e-mail: hc_ma@tju.edu.cn; Jiaji He, Tianjin University, Tianjin, Tianjin, China; e-mail: dochejj@tju.edu.cn; Yiqiang Zhao, Tianjin University, Tianjin, Tianjin, China; e-mail: yq_zhao@tju.edu.cn; Xiaolong Guo, Kansas State University, Manhattan, Kansas, United States; e-mail: guoxiaolong@ksu.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM 1539-9087/2025/10-ART157

<https://doi.org/10.1145/3768154>

ACM Reference Format:

Qizhi Zhang, Ya Gao, Haocheng Ma, Jiaji He, Yiqiang Zhao, and Xiaolong Guo. 2025. Boosting Cryptographic ICs' Side-Channel Resistance: A Formal Framework for Automatic Identification and Protection of Leaky Paths. *ACM Trans. Embedd. Comput. Syst.* 24, 6, Article 157 (October 2025), 25 pages. <https://doi.org/10.1145/3768154>

1 Introduction

Cryptographic algorithms are pivotal in modern information security, offering data encryption and identity authentication services. These algorithms are now integral to critical applications, including autonomous vehicles, electronic banking, mobile communication, and cloud computing. While these algorithms are mathematically robust and can withstand various direct attacks, their hardware implementations remain vulnerable to physical attacks, notably **side-channel analysis (SCA)** [1]. An SCA attack involves analyzing side-channel leakages to extract sensitive internal data, which is typically shielded from conventional cryptographic scrutiny. Such side channels encompass power consumption, timing delays, **electromagnetic (EM)** emanations, thermal outputs, optical leakages, and more. Notably, power and EM SCA attacks targeting cryptographic **integrated circuits (ICs)** have garnered significant attention in recent years.

Modern ICs feature a plethora of evaluation techniques and countermeasures to assess side-channel leakage and safeguard against SCA attacks. Conventional techniques for side-channel leakage assessment typically operate at the post-silicon stage, attempting an attack on the device. Recently, researchers have proposed various pre-silicon assessment methods, such as those presented in [2–5], enabling earlier leakage assessment in the digital circuit design flow. However, both post-silicon and pre-silicon methods depend on a considerable number of test vectors, rendering them inefficient and resource-intensive.

On the other hand, most of the prevailing hardware countermeasure schemes can be broadly categorized into two main techniques: hiding and masking. These techniques primarily target secret-dependent intermediate data, commonly referred to as sensitive variables. Such variables are often the primary sources of leakage during the execution of a cipher. The objective of hiding techniques is to diminish the correlation between the side-channel information and the sensitive variables. Conversely, masking techniques endeavor to obscure the internal computations, effectively decoupling the side-channel information from the sensitive variables [6]. Leveraging secret sharing, sensitive variables are divided into multiple shares and are then integrated into circuit operations independently. Nonetheless, these countermeasures not only lead to substantial overheads in power, area, and speed but also necessitate designers to possess an in-depth understanding of hardware security for accurate implementation.

The fundamental cause of this issue is that designers are unable to accurately identify the sources of side-channel leakage within the circuit, making it difficult to implement targeted security enhancement measures. Therefore, to address the aforementioned challenges, this article emphasizes the detection and obfuscation of leaky paths, the source of side-channel leakage in the circuit, at the gate-level design. These leaky paths are conduits through which information leakage, stemming from sensitive variables, permeates the design. Rather than safeguarding each individual vulnerable cell, countermeasures targeting leaky paths treat multiple vulnerable cells collectively for enhanced protection. This approach not only streamlines protection efforts by bypassing non-critical logic cells within a vulnerable module but also ensures a significant boost in security. Consequently, this results in a notable reduction in area, power, and performance overheads. For this purpose, we have developed a novel framework. Compared with our previous work [7], a more accurate and comprehensive theory for side-channel leaky path identification is proposed, along with an automated formal approach for identifying these paths. In the process of path identification,

we employ a hybrid approach, merging dynamic correlation analysis with formal-based security verification, to swiftly pinpoint all leaky paths. Subsequently, during path obfuscation, we harness hardware-centric solutions such as Boolean masking and precharge techniques to autonomously safeguard these identified paths. Notably, this framework seamlessly integrates into prevailing design flows and remains agnostic to any specific technology node.

The main contributions of the article are as follows.

- A side-channel tracking logic is proposed to characterize side-channel behavior in the netlist. The logic for all commonly-used logic cells is derived, which can be utilized in all “label tracking” based methods.
- In path obfuscation, logic transformation is used to automatically deploy hardware protection on leaky paths. Two obfuscation methods are employed, taking into consideration the balance between security and cost.
- A new EDA tool is developed to track side-channel leaky paths in the netlist. This tool can formally identify leaky paths and seamlessly integrates with the current EDA design flow.

The remainder of this article is structured as follows: Section 2 delves into current methods for leakage detection and countermeasures against SCA attacks. Side-channel tracking logic is introduced in detail in Section 3. Section 4 presents the whole proposed framework. Section 5 introduces the formal based tool which is designed for leaky paths identification. Section 6 presents both simulated and empirical results, comparing them with prior works. The article culminates in Section 8.

2 Background

2.1 Attack Model

Correlation Power Analysis (CPA) and **Correlation Electromagnetic Analysis (CEMA)** are considered as the main attack methods for cryptographic hardware implementation. This article assumes that attackers can collect power or EM data during circuit operation by some means. In addition, attackers can access input ports to get plaintext set of cryptographic circuit and have knowledge of the hardware functionality.

As an example, consider the CPA attack scenario: Assume that an attacker collects D power traces corresponding to different plaintexts, with each power trace comprising T power sample points. Consequently, a power data matrix of size $D \times T$ is collected. Utilizing the set of plaintexts, the attacker can construct an information leakage model. However, the exact key value remains unknown, necessitating the use of hypothesized key values for the model. With a 128-bit key, the number of potential keys is 2^{128} , rendering exhaustive computation infeasible. To mitigate this, the key search space is reduced by dividing the key into byte-sized groups. Upon establishing the leakage model, correlation analysis is employed to produce traces that reflect the correlation between the side-channel data and the leakage model. The correlation is computed as follows:

$$r_{i,j} = \frac{\sum_{d=1}^D (h_{d,i} - \bar{h}_i) \cdot (t_{d,j} - \bar{t}_j)}{\sqrt{\sum_{d=1}^D (h_{d,i} - \bar{h}_i)^2 \cdot (t_{d,j} - \bar{t}_j)^2}} \quad (1)$$

where D represents the number of power traces collected by attacker, $h_{d,i}$ represents information leakage model calculated by the d th plaintext and the i th guessed key, and $t_{d,j}$ represents the power trace of the d th plaintext in the j th sample point.

After Pearson correlation coefficients calculation, the correlation traces of guessed keys are formed. If the correlation trace between the information leakage model with correct key and side-channel numerical matrix has a distinct spike compared with other traces, it can be regarded as that the correct key is obtained by the attacker.

2.2 Leakage Detection Methods

With access to hardware design specifics, leakage detection at the pre-silicon stage is adept to pinpoint the origin of leakage. This empowers designers to fortify vulnerable designs before proceeding to fabrication. For instance, RTL-PSC initially computes the power profile of a **register-transfer level (RTL)** design and subsequently employs the Kullback–Leibler divergence combined with the success rate to detect susceptible blocks [2]. PLAN, on the other hand, conducts trace modeling on RTL codes and calculates **side-channel vulnerability (SCV)** metrics to identify leak-prone modules within designs [3]. SCRIPT first identifies the registers that cause side-channel leakage using information flow tracking, and then proposes a metric named SCV to assess side-channel leakage [4]. FORTIFY proposes **Signal Probability Correlation Factor (SPCF)** metric to evaluate power side-channel leakage in RTL or gate-level netlist stage [5]. ACA leverages gate-level simulation to rank design cells based on the magnitude of their information leakage contributions [8]. Additionally, CASCADE offers a comprehensive assessment of side-channel vulnerabilities within a netlist across all design phases [9].

Prior methodologies have been successful in pinpointing the origins of side-channel leakage down to specific modules or even individual logic cells. Such precision enables designers to deploy countermeasures with increased accuracy, targeting only the vulnerable segments of a design. However, by solely concentrating on vulnerable modules or cells, the broader implications of a leaky path are often overlooked. Within a leaky path, information leakage can cascade through a series of logic cells, rendering each cell within the path susceptible to SCA attacks. As this article will elucidate, countermeasures aimed at leaky paths circumvent the limitations inherent in block-level and cell-level strategies. Consequently, our focus in this article shifts from isolated vulnerable cells or modules to the comprehensive identification and protection of leaky paths that transmit sensitive information.

2.3 Leakage Mitigation Methods

A plethora of leakage mitigation techniques have been introduced, spanning various hardware description hierarchies, from comprehensive designs down to individual logic cells. Strategies such as noise injection [10], integrated voltage regulators [11], and current signature attenuation [12] are typically applied across entire circuits or specific submodules. These methods leverage auxiliary circuits to either introduce noise or attenuate the signal components of side-channel information. While these techniques bolster security, shielding an entire block amplifies challenges related to power, area, and performance. Another category of solutions encompasses secure logic styles, including **sense amplifier-based logic (SABL)** and **wave dynamic differential logic (WDDL)**. These styles aim to normalize or randomize side-channel behavior under varying conditions. Conventionally, design logic cells are globally substituted with their secure counterparts. Even in works like [8], where they are applied locally to identified vulnerable cells, the overhead becomes prohibitive as the number of vulnerable cells escalates. Ref. [13] introduced a **Probe-Isolating Non-Interference (PINI)** based method to automatically implement mask to hardware, however, similarly, the higher overhead limits its applicability for circuit designers. In the field of mask verification, works such as [14] and [15] have proposed verification and evaluation methods for masking schemes, providing guidance for more efficient security enhancement design.

Drawing inspiration from [3] and [8], our approach delves into the propagation of leakage due to the signal connectivity of susceptible logic cells. This allows us to categorize identified cells into distinct clusters, termed as leaky paths, and subsequently apply tailored countermeasures. The principle of masking stands out for its potential to obscure side-channel information within these leaky paths. Yet, traditional methods are often algorithm-dependent and necessitate intricate modifications across the entire circuit, leading to significant power and area overheads. To

circumvent these challenges, we introduce a framework designed to autonomously implement masking solutions directly on the hardware descriptions of leaky paths.

2.4 Netlist Level Security Check Using Label-Based Tracking Methods

Gate level information flow tracking (GLIFT) is a label based method for security checking in netlist stage [16]. In GLIFT, each data bit in hardware design is assigned by a label standing for the trust level. The label will be propagated to other data bits or ports according to information flow policy. Both value of data bits and security label are taken into consideration during the label propagation. Only data with high level label can be propagated to the trust portion in hardware design. Thus we can detect if sensitive information is leaked through observing value of the label of trust portion in hardware design.

Hu et al. derived the GLIFT logic formula for common logic cells and concluded fundamental theorem for GLIFT [17]. Several works have been published utilizing GLIFT to detect information leakage caused by Hardware Trojan or design fault. Ref. [18] introduces a total automatically framework to apply GLIFT to detect Hardware Trojan using Z3 SMT solver. Qin et al. proposed a theorem proving based GLIFT method for security verification in hardware design [19]. A unified formal model is proposed in [20, 21] which combines GLIFT policy and X-propagation rule to verify the security and integrity in hardware design. Moreover, GLIFT is also employed for timing side-channel security [22, 23]. However, GLIFT does not yet support power or EM side-channel assessment or detection in published works.

3 Side-Channel Tracking Logic

In the context of constructing side-channel leaky paths, we classify circuit cells into two categories: source cells and leakage cells. The source cell serves as the start of the paths, while leakage cells are those exhibiting side-channel behavior similar to the source cell. We will explain the location of the source cell in Section 4. To accurately identify leaky paths within the circuit, we propose a side-channel tracking logic metric in this section. Based on this metric, we can formally identify leakage cells through label propagation. To elaborate the metric in detail, side-channel propagation rule is firstly explained, and then the side-channel tracking label and its propagation logic are proposed and derived to formulate the propagation rule of side-channel leakage.

3.1 Side-Channel Propagation Rule

Assuming that the source cell of side-channel leakage has been located. Consider a specific logic cell as an example. This cell receives the **sensitive variable** v_x either from a preceding source cell or from an intermediate cell. Its logic function takes inputs $a_1, a_2, \dots, a_{n-1}, v_x$ and produces the **corresponding output** O_x . When the sensitive variable transitions from v_x to v_y , a state transition from O_x to O_y ensues. The state transitions $v_x \rightarrow v_y$ and $O_x \rightarrow O_y$ dictate the power profile of the logic cell and its preceding cell, respectively. If the logic cell is part of the leaky paths, its power profile should closely mirror that of the known preceding source cell or intermediate cell. Equation (2) encapsulates this relationship, defining it as the leakage attribute. It's worth noting that we overlook the minor discrepancies between the power profiles of state transitions $0 \rightarrow 1$ ($0 \rightarrow 0$) and $1 \rightarrow 0$ ($1 \rightarrow 1$). This implies that there is an input state such that an intermediate cell will propagate state transitions of sensitive variables to its output net.

$$\exists(a_1, a_2, \dots, a_{n-1}), |O_c - O_p| = |v_c - v_p| \quad (2)$$

3.2 Side-Channel Tracking Label

Drawing inspiration from GLIFT [17, 18], a label-based information tracking method, we propose the concept of a side-channel tracking label to describe side-channel tracking logic more mathematically.

Table 1. Truth Table of Two-Input AND Gate

#	A	B	Y
1	0	0	0
2	0	1	0
3	1	0	0
4	1	1	1

A side-channel tracking label is a bit-level label associated with a data bit in the original circuit. We use the subscript s to denote the label. A label is referred to as “tainted” when set to the logical value 1, implying that the data bit is side-channel-sensitive and should be tracked throughout the system. If the output exhibits similar side-channel behavior as the tainted input, it is marked as tainted. We consider a two-input AND gate as an example for a better understanding of the side-channel label. We observe the output when the input changes to discern if the output exhibits the same side-channel behavior as any of the inputs.

In the truth table (Table 1), A, B, and Y are used to denote inputs and output, respectively. Considering rows 1 and 2 of the truth table, when input A changes from 0 to 0 and input B changes from 0 to 1, the output Y changes from 0 to 0. We can conclude that the output Y exhibits a power profile or side-channel behavior similar to that of input A, under the condition that B changes from 0 to 1.

$$(A \ 0 \rightarrow 0 \&\& B \ 0 \rightarrow 1 \&\& A_s = 1) \Rightarrow Y_s = A_s \quad (3)$$

Now we consider the state change between rows 2 and 3. Both inputs have changed: A changes from 0 to 1, and B changes from 1 to 0. However, the output Y remains the same. It means that the power profile or side-channel output behavior is not similar to that of either input. In this event, irrespective of which input signal is sensitive, the logic cell cannot propagate side-channel behavior even if both inputs are tainted.

$$(A \ 0 \rightarrow 1 \&\& B \ 1 \rightarrow 0) \Rightarrow Y_s = 0 \quad (4)$$

Thus, the properties of the side-channel tracking label are summarized as below:

Property (1): The propagation of side-channel label is not only related to labels of inputs but also to values and state changes of inputs;

Property (2): When all inputs are untainted, the output is always untainted;

Property (3): When all inputs are tainted, the output is not necessarily tainted;

Property (4): The inverter changes the logic value of the original circuit but cannot change the logic value of the side-channel label.

3.3 Deriving Side-Channel Tracking Logic

Based on the above properties, the side-channel tracking logic of common logic cells are derived according to their truth tables and some basic laws of logical operations.

(1) Side-channel tracking logic of AND gate: Take AND-2 gate as an example, the original logic of AND-2 is $Y = A \cdot B$. All the changing conditions are studied to derive the side-channel tracking logic of AND-2. To completely describe the side-channel tracking logic, the changing label is introduced. We use the subscript c to represent the changing situation of a data bit. For instance, if $A_c = 1$, it means the data bit A will change its value in next state. Referring the logic change between row1 and row2 in Table 1 we discussed above, the side-channel tracking logic is shown in Equation (5).

$$Y_s = (\bar{A} \cdot \bar{A}_c \cdot B_c) \cdot A_s \quad (5)$$

According to commutative law of logic execution, $Y = A \cdot B = B \cdot A$. We can easily get the expression of the logic change between row1 and row3 based on Equation (5) as below:

$$Y_s = (\bar{B} \cdot \bar{B}_c \cdot A_c) \cdot B_s \quad (6)$$

When both inputs change from 0 to 1 or 1 to 0, output will change from 1 to 0 or 0 to 1, respectively. Under this condition, output is tainted so long as one input is tainted.

$$Y_s = (A_c \cdot B_c \cdot (A \odot B)) \cdot (A_s + B_s) \quad (7)$$

Referring the logic of row2 and row3 and the property(4) we discussed above, when two inputs change from 0 to 1 and from 1 to 0 respectively, the output stays 0. No matter which input is tainted, the output will not be tainted. The side-channel tracking logic is obtained.

$$Y_s = 0 \quad (8)$$

Similar with the situation we discussed in equation(5), when input A stays 1 and input B changes, the output will be tainted if input B is tainted. The side-channel tracking logic can be concluded as

$$Y_s = (A \cdot \bar{A}_c \cdot B_c) \cdot B_s \quad (9)$$

Similarly, it can be obtained from the commutative law that when the input B stays 1 and input A changes, the side-channel tracking logic is

$$Y_s = (B \cdot \bar{B}_c \cdot A_c) \cdot A_s \quad (10)$$

At last, if both inputs stay their logic value, the output will stay its logic value too. The output will be tainted so long as one of any input is tainted. The side-channel tracking logic can be concluded as

$$Y_s = (\bar{A}_c \cdot \bar{B}_c) \cdot (A_s + B_s) \quad (11)$$

Based on the results of the above classification discussion, the side-channel tracking logic of AND-2 can be conclude as

$$\begin{aligned} Y_s = & (\bar{A} \cdot \bar{A}_c \cdot B_c + B \cdot \bar{B}_c \cdot A_c) \cdot A_s + \\ & (\bar{B} \cdot \bar{B}_c \cdot A_c + A \cdot \bar{A}_c \cdot B_c) \cdot B_s + \\ & (A_c \cdot B_c \cdot (A \odot B) + (\bar{A}_c \cdot \bar{B}_c)) \cdot (A_s + B_s) \end{aligned} \quad (12)$$

Further, the input of AND-2 can not only be logic variables but also functions. Thus, the logic of multi-input AND gates can be substituted into Equation (12) according to the associative law of logical operations to get the side-channel tracking logic.

(2) Side-channel tracking logic of OR gate: Take OR-2 gate as an example. The logic expression of OR-2 gate is $Y = A + B$. Based on De Morgan's Law, the logic of OR gate can be rewrite as

$$Y = \overline{\bar{A} \cdot \bar{B}} \quad (13)$$

According to **Property (4)**, assuming that there is an AND-2 gate such that $G = \bar{A} \cdot \bar{B}$, we can obtain the conclusion $Y_s = G_s$. Based on Equation (12), it can be inferred that the side-channel tracking logic of OR-2 is

$$\begin{aligned} Y_s = & (A \cdot \bar{A}_c \cdot B_c + \bar{B} \cdot \bar{B}_c \cdot A_c) \cdot A_s + \\ & (\bar{B} \cdot \bar{B}_c \cdot A_c + \bar{A} \cdot \bar{A}_c \cdot B_c) \cdot B_s + \\ & (A_c \cdot B_c \cdot (A \odot B) + (\bar{A}_c \cdot \bar{B}_c)) \cdot (A_s + B_s) \end{aligned} \quad (14)$$

Similarly with AND-2 gate, the side-channel tracking logic of OR-2 can also be extended to multi-input OR gate according to the associative law of logical operations.

Table 2. Change Label Logic of Logic Cells

Logic cell types	Change label logic
AND-2	$Y_c = (A_c \cdot B + A \cdot B_c) \cdot (A_c \oplus B_c) + (A_c \cdot B_c) \cdot (A \odot B)$
OR-2	$Y_c = (\bar{A}_c \cdot B + A \cdot \bar{B}_c) \cdot (A_c \oplus B_c) + (A_c \cdot B_c) \cdot (A \odot B)$
XOR-2	$Y_c = (A_c \oplus B_c)$
INV	$Y_c = A_c$
DFF	$Y_c = A_c$

(3) Side-channel tracking logic of XOR gate: Take XOR-2 gate as an example, the logic expression of XOR-2 gate is $Y = A \otimes B$ which can be rewrite using *and or* logic as below:

$$Y = (A \cdot \bar{B}) + (\bar{A} \cdot B) \quad (15)$$

It can be considered as a combination of an AND-2, an NAND-2, and a OR-2 gates. According to **Property (4)**, NAND-2 has the same side-channel tracking logic with AND-2. Thus, the side-channel tracking logic can be derived based on Equation (12) and Equation (14).

$$Y_s = \bar{B}_c \cdot A_s + \bar{A}_c \cdot B_s \quad (16)$$

(4) Side-channel tracking logic of INV and DFF: According to **Property (4)**, the inverter can't change the value of side-channel label, so that the side-channel tracking logic of INV is

$$Y_s = A_s \quad (17)$$

As a logical cell for storing and propagating data, DFF does not have the function of changing logical values. So that the side-channel behavior of output data is always same as the input data. The side-channel tracking logic of DFF is also as described in Equation (17).

3.4 Change Label Propagation Logic

To precisely track the side-channel taint propagation, the propagation of the change label needs to be tracked in the meantime. Similar to side-channel tracking labels, the propagation rule of change labels can be concluded from the truth table as well. A two-input AND gate is discussed as an example below.

(1) One input is changed: If B keeps unchanged 0, $Y_c = 0$ and if B keeps unchanged 1, $Y_c = A_c$. Similarly, if A keeps unchanged 0, $Y_c = 0$ and if A keeps unchanged 1, $Y_c = B_c$. It can be described with logic expression as below:

$$Y_c = (A_c \cdot B + A \cdot B_c) \cdot (A_c \oplus B_c) \quad (18)$$

(2) Two inputs are changed: Only when A and B are the same logic value and change together, the output Y can be changed. It can be described with logic expression as below:

$$Y_c = (A_c \cdot B_c) \cdot (A \odot B) \quad (19)$$

The change label propagation rule of 2-AND can be concluded as

$$Y_c = (A_c \cdot B + A \cdot B_c) \cdot (A_c \oplus B_c) + (A_c \cdot B_c) \cdot (A \odot B) \quad (20)$$

The change label propagation rule of common types of logic gates can be induced in the same way, and logic expression is summed up in Table 2.

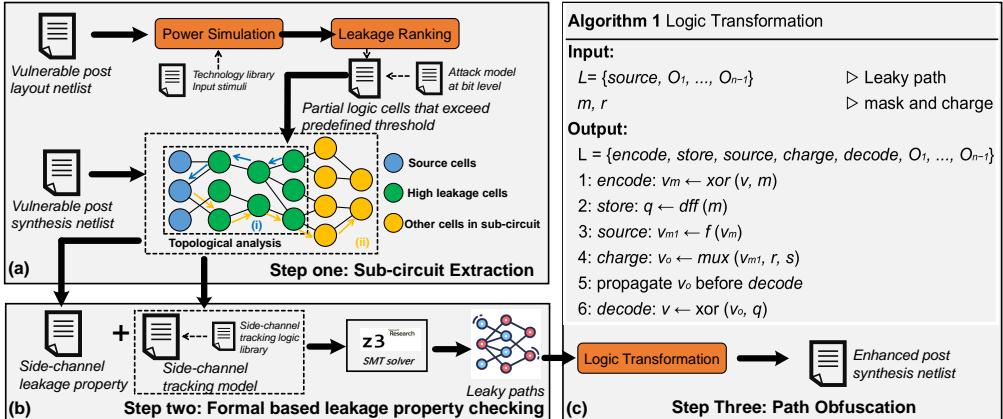


Fig. 1. The overall workflow of the proposed framework.

Based on the theory of side-channel tracking logic, we can convert circuit design to formal constraints and track side-channel tracking labels. Property checking will be performed in the SMT solver Z3 to extract the whole leaky path.

4 Methodology

In this section, we introduce the proposed framework. Figure 1 illustrates the overall workflow, which consists of sub-circuit extraction, formal-based leakage property checking, and path obfuscation. Firstly, source cells of leaky paths are identified through dynamic correlation analysis, followed by the extraction of a *sub-circuit*. This *sub-circuit*, smaller in scale than the entire netlist and containing leaky paths, is obtained through topological analysis. Secondly, a formal model is constructed based on the theory discussed in Section 3. Leaky paths are then detected via property checking in the SMT solver. Lastly, a carefully designed masking method is applied to enhance the circuit's side-channel security. The first two steps collectively complete the path identification process.

4.1 Sub-Circuit Extraction

(1) Dynamic Correlation Analysis: Drawing inspiration from [8], dynamic correlation analysis ranks logic cells based on the magnitude of their information leakage. This ranking is achieved by measuring the correlation between the dynamic power traces of logic cells and established leakage models. Conventionally, designers classify cells with information leakage exceeding a certain threshold as vulnerable and subsequently apply mitigation techniques. However, when formulating comprehensive leaky paths, dynamic correlation analysis faces challenges. Accurately yet efficiently collecting dynamic traces can be challenging in constrained experimental settings. Moreover, preset thresholds might overlook logic cells that contribute collectively to side-channel leakage. Such constraints can lead to either inflated costs or incomplete path identification outcomes. To address this, dynamic correlation analysis is tailored to select a subset of the most vulnerable cells based on their leakage contributions, as depicted in Figure 1(a). The setup of the threshold is described in Section 6.1.

Initially, the dynamic power of each logic cell is gathered via gate-level power simulations. Using a set of stimuli, we conduct functional simulations on the post-layout netlist, which is back-annotated with precise delay data. The switching activities of all logic cells are logged over a specified duration. This data, in conjunction with the technology library, design netlist, and parasitic information, is fed into Primetime PX for temporal power analysis.

Subsequently, the information leakage of each logic cell is quantified, employing the maximum Pearson correlation coefficient ρ as the leakage metric. As expressed in Equation (21), this metric C reveals the relationship between power traces P and the leakage model L . Here, the leakage model represents the Hamming weight or Hamming distance of sensitive variables per bit v . The onus is on the designer to choose which sensitive variables require protection, such as cryptographic algorithm keys. The framework then ranks logic cells within the netlist from the highest to the lowest leakage metric. Based on a predetermined threshold, we identify a subset of logic cells responsible for the most significant information leakage by examining all data bits.

$$C = \max(|\rho(P, L)|) \quad (21)$$

For a single leaky path, the information leakage instigated by sensitive variables originates from the source cell and traverses multiple intermediate cells before reaching irrelevant ones. Typically, the source cell is the primary leaker of information. Hence, we can pinpoint source cells within the subset of logic cells derived from the aforementioned process. As illustrated in Figure 1(b), topological analysis facilitates this identification. Concurrently, attribute inspection extends to subsequent intermediate cells to formulate complete leaky paths.

(2) Topological Analysis: Given that the post-synthesis netlist comprises multiple modules, topological analysis initially segregates it into distinct module files. Subsequently, the inherent connectivity of each module is represented using two dictionaries. The first dictionary, termed **CELL**, enumerates signal nets entering and exiting each logic cell. The second dictionary, dubbed **NET**, logs the logic cells connected upstream and downstream of each signal net. Leveraging **CELL** and **NET**, we can efficiently deduce either the preceding or succeeding network relative to a specific logic cell in a step-wise manner. Following this, this framework reintegrates the subset of logic cells into the post-synthesis netlists and constructs preceding networks confined to this subset. The cells positioned at the forefront of these networks are identified as source cells within the leaky paths. Sensitive variables commence their propagation from these source cells, resulting in power profiles that exhibit strong correlation with them.

After locating the source cells, the topological analysis is reversely applied again to extract the portion of the net-list that is related to the source cells. A part of the netlist that starts from the source cell and end of the output is delivered. All signals that come from logic cells before the source cell are set as inputs of the part of the netlist. The portion of the netlist is named *sub-circuit*, which is to be modeled for side-channel leakage path tracking.

4.2 Formal Based Leakage Property Checking

We proposed a formal-based property checking method to extract leaky paths by realizing the side-channel tracking logic of *sub-circuit* in the SMT solver. The procedure of leaky paths extraction is shown in Figure 1(b). The netlist of *sub-circuit* is input to a model generation parser which converts hardware design to a formal model. Based on the **Property (1)** we discussed in Section 3.2, the model not only includes side-channel tracking logic and change logic of *sub-circuit*, but also includes its original functions. We name these side-channel tracking logic models as S , change the logic model as C , and the original function model is called O . Thus we can define the formal model M of *sub-circuit* as Equation (22):

$$M := O \wedge S \wedge C \quad (22)$$

After modeling procedure, the model M is delivered to the SMT solver. The property to track side-channel label propagation is defined and input to the SMT solver as well. We denote the property as P . Take AND-2 as an example, assuming that input A is high side-channel sensitivity and input B is low side-channel sensitivity which leads to logic value 1 in A_s and logic value 0 in

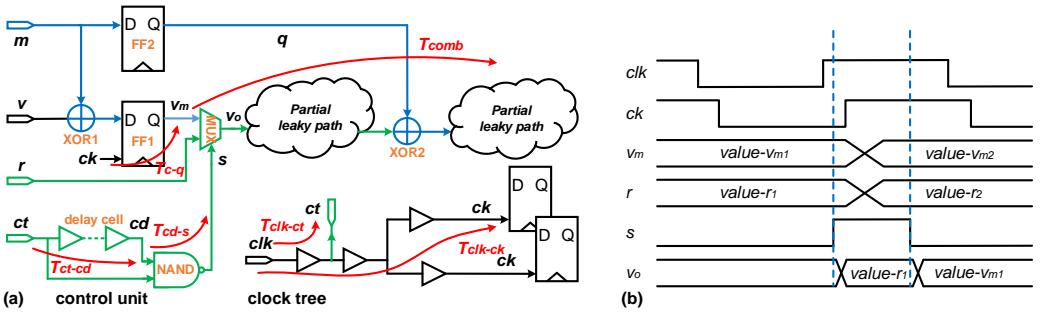


Fig. 2. The hardware scheme of protection solutions, including Boolean masking (blue) and random precharge (green).

B_s . The property P and the whole representation R in the SMT solver can be shown as follows:

$$P := (A_s == 1) \wedge (B_s == 0) \quad (23)$$

$$R := M \wedge P \quad (24)$$

Then the SMT solver automatically solves R and gives solutions. The solution contains all signals and their labels of *sub-circuit*. If the side-channel tracking label of which signal is 1 in solution, it means that the signal is side-channel leakage high sensitively. And the logic cell of which the signal is the output will be considered as a part of leaky paths. Through solution analysis, the complete leaky paths are constructed.

4.3 Path Obfuscation

Upon identifying the leaky paths, an efficient and straightforward countermeasure, such as Boolean masking or random precharge, is automatically chosen to locally safeguard them, as depicted in Figure 1(c). Logic transformation facilitates the application of these protection measures to the post-synthesis netlist.

(1) Logic Transformation: The finalized hardware architecture is presented in Figure 2(a). It encompasses the original logic (represented in black), Boolean masking (in blue), and random precharge (in green). Taking a single path as an example, Boolean masking initiates at the source cells, typically involving sequential logic. An XOR gate encodes the sensitive variable v using uniformly random masks m (Line 1). The masked variable V_m traverses the leaky path, ensuring the entire data flow remains obfuscated (Lines 3 and 5). An additional XOR gate is introduced prior to irrelevant cells to decode the masked variables (Line 6). Consequently, a **flip-flop (FF)** is essential to retain and relay the masks. The proposed framework empowers users to specify the quantity of masks and their corresponding FFs (Line 2).

However, as discussed in Section 3.3, introducing an additional XOR gate to encode the sensitive variable could potentially create another leakage point in the logic cell. To mitigate the side-channel leakage introduced by the added XOR gate, we employ the random precharge method.

A 2-to-1 MUX gate is integrated between the source cell and intermediate cells (Line 4). The select input s dictates which data input (either masked variable V_m or random charge r) is relayed to the output. A pivotal consideration for successful deployment is the regulation of the protection mechanism. The framework incorporates a control unit, consisting of a NAND gate and several delay cells. This unit receives the control signal ct proximate to the clock source and emits a brief pulse signal s to the MUX gate. When s retains a logic high state, the random charge is conveyed. In other scenarios, the source cell's values are transmitted.

For every leaky path, the hardware architecture for the obfuscation method is implemented as presented in Figure 2(a).

Figure 2(b) showcases a timing diagram of pertinent signals. As such, the configurations of the control signal and delay cells significantly influence both security and performance, necessitating adherence to specific conditions:

$$T_{clk-ct} + T_{comb} + T_{cd-s} > T_{clk-ck} + T_{c-q} + T_{hold} \quad (25)$$

$$T_{clk-ct} + T_{cd-s} < T_{clk-ck} + T_{c-q} \quad (26)$$

$$T_{clk-ct} + T_{ct-ed} + T_{cd-s} > T_{clk-ck} + T_{c-q} \quad (27)$$

$$T_{clk-ct} + T_{ct-ed} + T_{cd-s} + T_{comb} < T_{clk-ck} + T_{cycle} - T_{setup} \quad (28)$$

where T_{cycle} , T_{setup} , and T_{hold} denote the clock period, setup time, and hold time, respectively. Types of T_{a-b} denote the delay from signal a to signal b . T_{comb} is the total delay of combinational logic behind FF1. Equations (25) and (26) delineate the temporal window during which the positive edge of the pulse signal s is expected to occur. An early arrival of the positive edge results in random charges being forwarded to the subsequent level of FFs, leading to incorrect outcomes. Conversely, if the positive edge arrives late, random charges may not effectively influence certain leaky paths. The timing constraints for the arrival of its negative edge are specified by Equations (27) and (28). It is imperative to ensure that there are no sampling errors when relaying values from source cells. Additionally, it's crucial to circumvent setup time violations that might arise due to added delays. By resolving the aforementioned equations, we can ascertain the position of ct and the requisite number of delay cells. This, in turn, imposes a new timing requirement during the placement and routing phase. The framework revises the timing constraint, augmenting the clock uncertainty by a factor of $|T_{clk-ct} + T_{cd-s} - T_{clk-ck}|$ delays.

(2) Security Analysis: For each sensitive variable bit, the state transition $v_1 \rightarrow v_2$ takes place prior to protection. With the application of the path obfuscation method, this transition is transformed into $r \rightarrow (v_1 \oplus m)$ for combinational logic based on random precharge method and $(v_1 \oplus m) \rightarrow (v_2 \oplus m)$ for sequential logic based on Boolean masking method. Given that both the mask m and charge r are uniformly randomized in each cycle, the state transition is effectively shuffled, leading to the obfuscation of dynamic power. Furthermore, these state transitions induce currents across metal wires, resulting in the emission of EM emanations. Consequently, the obfuscated state transitions also fortify the design against potential EM SCA attacks.

5 Tool Design

An automated tool has been developed for side-channel tracking model generation and leakage path extraction. It initially translates the netlist design of *sub-circuit* into Z3-SMT solver constraints, following the logic discussed in Section 3.2. Subsequently, the side-channel tracking property is established according to the source cell identified in Section 3.1. Incremental solving is then applied to identify as many solutions as possible that satisfy the leakage property while keeping time consumption to a minimum. Lastly, the leaky paths are extracted in conjunction with the logic cells. The tool, written in Python, comprises four modules, as depicted in Figure 3.

5.1 Model Generation Parser

We developed an automatic parser for converting a netlist design to Z3-SMT solver constraints. The parser has three blocks, as shown in Figure 3. The input of the parser is the netlist design of *sub-circuit*, while the output is a formal model for side-channel tracking. The code analysis part first interprets the input *sub-circuit* design. Wires and registers are extracted as symbolic variables

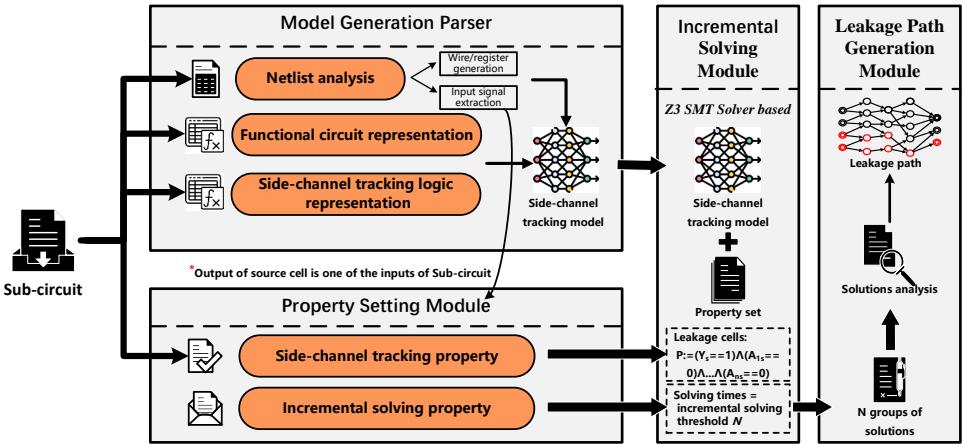


Fig. 3. The block diagram of the designed tool.

Original netlist	Wire Declaration & Functional Representation	Side-channel Tracking Logic Representation
<pre> input N1, N2, N3, N6, N7; output N22; wire N1, N2, N3, N6, N7, N10, N11, N16, N19, N22; NAND2X1 U1 (.A(N1), .B(N3), .Y(N10)); NAND2X1 U2 (.A(N3), .B(N6), .Y(N11)); NOR2X1 U3 (.A(N2), .B(N11), .Y(N16)); OAI221XL U4 (.AO(N7), .A1(N17), .B0(N10), .B1(N16), .CO(N19), .Y(N22)); </pre>	<p>Newly added wires</p> <pre> N1 = BitVec('N1', 1) N1_c = BitVec('N1_c', 1) N1_s = BitVec('N1_s', 1) ... N22_3 = BitVec('N22', 1) N22_3_c = BitVec('N22_3_c', 1) N22_3_s = BitVec('N22_3_s', 1) </pre> <p>split</p> <pre> N10 == ~N1 & N3, N11 == ~N3 & N6, N16 == ~N2 N11, N22 == ~N22_0 & N22_1 & N22_3, N22_0 == N7 N11, N22_1 == N10 N16, N22_3 == N19, </pre>	<p>N10_c == N3 & N1_c N1 & N3_c & N1_c & N3_c N1_c & N3_c & ~N1 & N3,</p> <p>N10_s == (~N1 & ~N1_c & N3_c N3 & ~N3_c & N1_c) & N1_s (~N3 & ~N3_c & N1_c N1 & ~N1_c & N3_c) & N3_s (N1_c & N3_c & ~N1 & N3) (~N1_c & ~N3_c) & (N1_s N3_s),</p> <p>N11_c == N6 & N3_c N3 & N6_c & N3_c & N6_c N3_c & N6_c & ~N3 & N6,</p> <p>N11_s == (~N3 & ~N3_c & N6_c N6 & ~N6_c & N3_c) & N3_s (~N6 & ~N6_c & N3_c N3 & ~N3_c & N6_c) & N6_s (N3_c & N6_c & ~N3 & N6) (~N3_c & ~N6_c) & (N3_s N6_s),</p> <p>N22_3_s == N19_s,</p>

Fig. 4. A model generation example for the parser.

in the Z3-SMT solver. Especially, the input signals of *sub-circuit*, which are utilized to generate the corresponding properties, are declared in the meantime. The Z3 constraints describing the original logic of *sub-circuit* are produced in the functional circuit representation generation block. *Sub-circuit* is input to the side-channel tracking logic representation module as well, which is used to generate a side-channel tracking model based on the logic discussed in Section 3.2. As shown in Figure 4, specifically, every logic cell in the netlist is taken into consideration to generate its corresponding functional representation and side-channel tracking logic representation. In particular, in order to reduce the complexity of the model and improve solution efficiency, for logic cells with multiple inputs and complex functions like *OAI221XL* shown in Figure 4, the parser split it to basic logic cells such as AND-2 and OR-2 at first and representations are generated then. Correspondingly, newly added wires are declared as well. After all of that, the model is integrated from the above three parts.

5.2 Property Settings

There are two parts to property setting modules. Firstly, to extract leakage paths in the generated model, leakage property is proposed according to the scl and input signals. Secondly, the output signal of the source cell is labeled as tainted, which means it's side-channel leakage sensitive. To

ensure the label propagation can't be influenced by irrelevant signals, the labels of other inputs should be set as untainted.

$$Y_s == 1 \wedge A_{1s} == 0 \wedge \dots \wedge A_{ls} == 0 \quad (29)$$

In Equation (29), Y represents the output of the source cell as well as one of the inputs and A_l represents other inputs of *sub-circuit*. We denote Equation (29) as P_{lk} . To obtain more solutions that meet P_{lk} , the incremental solving property is set subsequently. The predefined threshold is set as the core character of the property.

$$M[n] != M[n - 1], n = 1, 2, 3, \dots, N - 1 \quad (30)$$

In Equation (30), M represents the model in the Z3-SMT solver and n represents the solving times. $M[n]$ means that solution get in the n th solving. Solving times N is set as a threshold to stop solving.

5.3 Incremental Solving

Because of the characteristic of Z3 that it stops when it finds a solution, to achieve obtaining more solutions in Z3, the incremental solving module is exploited. Incremental property described in Equation (30) is set in the property set module. In more detail, any variable in the solution that is different from the last solution can lead to a new solution.

$$\exists v_i \text{ s.t. } M[n](v_i) != M[n - 1](v_i), i = 1, 2, 3, \dots, m \quad (31)$$

In this equation, v_i represents the i th variable, and m represents the total number of variables in model M . Furthermore, all variables in model are signals in *sub-circuit* design and its logic value can be determined by inputs. Thus, inputs of *sub-circuit* can be utilized to realize incremental solving. Equation (31) can be modified as below:

$$\exists a_i \text{ s.t. } M[n](a_i) != M[n - 1](a_i), i = 1, 2, 3, \dots, l \quad (32)$$

A more specific description for incremental solving is delivered where a_i represents the i th input, and l represents the total number of inputs. In these two equations, $l \ll m$, the conditions to realize incremental solving are decreased while constraints for model M stay the same, in fact. We denote Equation (32) as P_{is} .

The whole solving process is shown as Algorithm 1. The inputs include model M , property P_{lk} , P_{is} and solving threshold N . Firstly, M and P_{lk} are input to the Z3-SMT solver and solved. If there is a solution that makes M satisfy P_{lk} , output the solution $M[n]$ and add P_{is} to the solver, n means the solving times. Solving process is total-automatically applied by the Z3-SMT solver. The solver gives different solutions until solving times meets the threshold N , or there is no more solution that suits P_{lk} . In the end, the solution set S contains N group solutions is collected.

5.4 Leakage Path Generation

The solution obtained from the incremental solving module shows the logic value of all signals as well as their labels. From that, firstly, all signals are extracted whose side-channel tracking labels are tainted. They are treated as variations that inherit the behavior of side-channel. And then, logic cells, of which signals with tainted labels are outputs, are traced back. After that, leaky paths are manifested as a network according to topological analysis.

6 Experimental Results

In this section, we use 128-bit AES circuit (denoted as AES-128) and PRESENT circuit with 64bit blocks and 80bit keys (denoted as PRESENT) to validate the efficacy of the framework on side-channel leaky paths identification and protection. At first, formal verification is applied to identify leaky paths in encryption circuits. Next, masking methods are implemented in the netlist design of

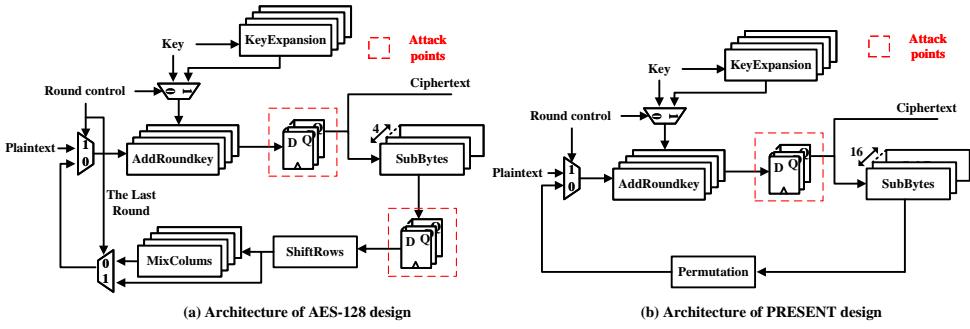


Fig. 5. The architectures of AES-128 design and PRESENT design.

ALGORITHM 1: Incremental solving process**Input:** Model M , Property P_{lk} and P_{is} , Solving Threshold N **Output:** Solutions S

```

1: Input  $M, P_{lk}$  to Z3
2:  $n \leftarrow 0$ 
3: if  $n < N$  then
4:   if sat in Z3 then
5:      $S[n] \leftarrow M[n]$ 
6:     Add  $P_{is}$  to  $M$ 
7:      $n \leftarrow n + 1$ 
8:   else
9:     return  $S$ 
10:  end if
11: else
12:   return  $S$ 
13: end if

```

encryption circuits, and simulation results are exploited to evaluate the hardened circuits in ASIC implementation.

6.1 The Framework Configuration

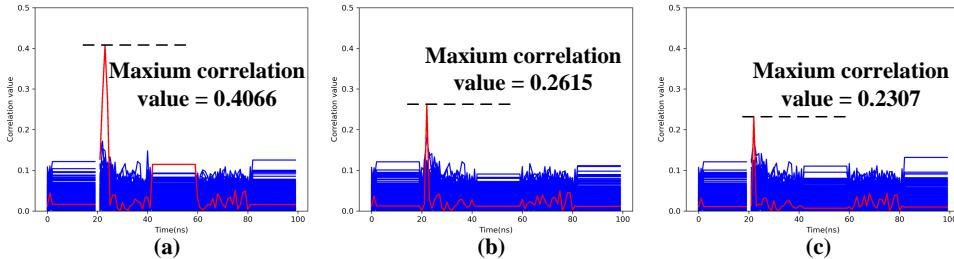
We implemented the encryption designs using logic synthesis and a 180 nm CMOS technology. For AES-128, the resulting post-synthesis netlist comprises 10 modules and 10,083 logic cells, processing 4 bytes of plaintext data per clock cycle. We configured the supply voltage and clock frequency to 1.8 V and 25 MHz, respectively. For PRESENT, the resulting post-synthesis netlist comprises 23 modules and 2,662 logic cells, processing 8 bytes of plaintext data per clock cycle. The supply voltage and clock frequency are configured to 1.8 V and 50 MHz. Subsequently, the designs underwent placement and routing processes to produce the final physical layout. In parallel, we gathered essential files, encompassing the post-layout netlist, delay specifics, and parasitic data. The architectures of these two encryption circuits are illustrated in Figure 5. The attack points are also marked in Figure 5.

In line with the methodology delineated in Section 4, we aimed to identify and obfuscate leaky paths to fortify the encryption design. The configuration parameters of our framework are detailed in Table 3.

In statistical analysis, a correlation coefficient value greater than 0.95 is generally considered to indicate a significant correlation between two sets of data. Yao et al. [8] define reference values at

Table 3. Parameters of the Framework for the Experiment

Input stimuli	Leakage criterion	Incremental threshold	Masks and Charges
1000	0.95	10	32

Fig. 6. (a) CPA result for AES-128. (b) CPA result after removing $Cell_p$. (c) CPA result after removing $Cell_f$.

99%, 95%, and 90% for the Pearson Correlation Confidence Interval. Coefficients within this range are viewed as particularly relevant for further analysis. Based on this analogy, a threshold of 0.95 is established for the leakage metric C in dynamic correlation analysis. This threshold is chosen to effectively balance computational overhead while ensuring accurate localization of the source cell.

The distribution of leakage metric C is analyzed. Take the 0th bit and the 31st bit of *Key* in AES circuit as examples. For the 0th bit of *Key*, only about 10 logic cells whose information leakage metric C is greater than 0.95 and information leakage metric C for other logic cells are all less than 0.3. For the 31st bit of *Key*, only 16 logic cells whose information leakage metric C is greater than 0.95, about 26 logic cells whose information leakage metric C is greater than 0.2 and less than 0.7, and information leakage metrics C of other logic cells are all less than 0.2. In order to reduce the time overhead of analysis as much as possible, we set the threshold as 0.95 to extract logic cells with high risk of leakage information.

During the dynamic correlation analysis, in AES-128, we identified 1,082 partial logic cells, given 1,000 input stimuli and a leakage criterion exceeding threshold. A total of 262 resource cells are located after topological analysis. Total of 1411 logic cells compose complete leaky paths after formal-based leakage property checking with single counting, accounting for 13.99% of the entire circuit.

Compared to Pathfinder [7], which identified 1,072 leakage cells with single counting, an additional 339 leakage cells were detected. To validate the effectiveness of the additionally detected leakage cells, we removed the power consumption of the leakage cells identified by Pathfinder (marked as $Cell_p$) and leakage cells identified by framework proposed in Section 4 (marked as $Cell_f$), and then conducted CPA attacks with 1000 power traces separately. As shown in Figure 6, compared to the CPA attack results without removing the power consumption of the leakage cells, removing the power consumption of the leakage cells significantly reduces the peak of the correct key's corresponding trace in the CPA analysis. The maximum correlation value decreased from 0.4066 to 0.2615. Furthermore, compared to removing the power consumption of $Cell_p$, the CPA analysis results show that the peak further decreases after removing the power consumption of $Cell_f$. And the maximum correlation value decreased from 0.2615 to 0.2307. This indicates that $Cell_f$ contributes more to the side-channel leakage of the circuit than $Cell_p$.

In the PRESENT circuit design, a total of 64 source cells are located. A total of 192 cells are identified as leakage cells that compose leaky paths, representing 7.21% of the entire circuit. In path obfuscation, **pseudo-random number generators (PRNG)** are inserted to produce 32 random masks and random charges, respectively. Additionally, MUXes are inserted to implement

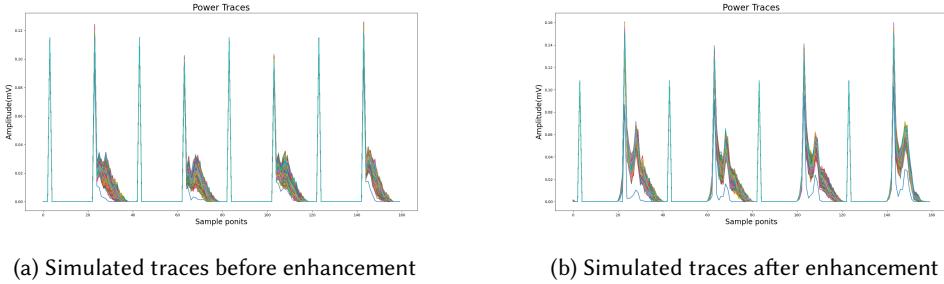


Fig. 7. Simulated traces for AES-128.

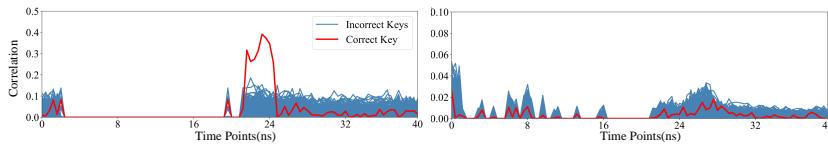


Fig. 8. Correlation traces as a function of time points obtained from unprotected (left) and protected (right) design for AES-128.

the combination of Boolean masking and random precharge. It takes about 18.54 min for dynamic correlation analysis in AES design and about 6.53 min in PRESENT design, about 1 s for one leaky path identification based on formal solving, and about 0.1 s - 0.2 s for masking implementation.

6.2 Analysis of Simulated Results

We conducted time-based power simulations on post-layout netlists using the Primetime PX tool to assess security. Simulated power traces before and after enhancement are shown as Figure 7(a).

For the Boolean masking and random precharge masking methods, we collected a total of 100,000 dynamic traces for both unprotected and protected AES-128 design, and a total of 20,000 dynamic traces for both unprotected and protected PRESENT design. Subsequently, we executed CPA attacks during the time intervals corresponding to the SubBytes operation of the initial four bytes/bits. In these CPA attacks, the key recovered with the highest Pearson correlation coefficient, denoted as ρ_{max} , suggests the most probable correct hypothesis. As the trace count surpasses a specific threshold, the ρ_{max} for the correct hypothesis consistently surpasses those of incorrect guesses. This threshold, commonly referred to as the **measurement to disclosure (MTD)**, signifies the minimum traces required for key disclosure. We employed both CPA and MTD metrics to gauge the resilience of designs against SCA attacks.

Figure 8 depicts the temporal trends of the maximum correlation ρ_{max} for all key hypotheses. For the unprotected design, the peak correlation of the accurate key nears 0.391 (top sub-figure). These results suggest a strong correlation between its side-channel behaviors and the leakage models employed by the attacker. Consequently, the attacker discerns the correct key using merely 84 traces (Figure 9 (left)). In contrast, for the fortified design, hardware countermeasures diminish the peak correlation to a mere 0.018 (bottom sub-figure). This reduction causes the correlation trace of the correct key to be overshadowed by those of incorrect key hypotheses. As a result, even with 100,000 traces, the attacker fails to extract the correct key (Figure 9 (right)). From a power side-channel resistance standpoint, the hardware countermeasures yield an enhancement of at least 1,190-fold.

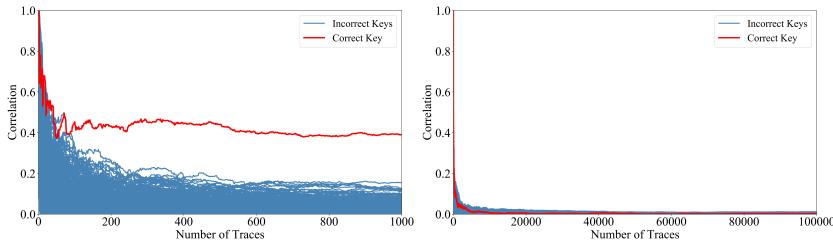


Fig. 9. Correlation traces as a function of stimuli number obtained from unprotected (left) and protected (right) design for AES-128.

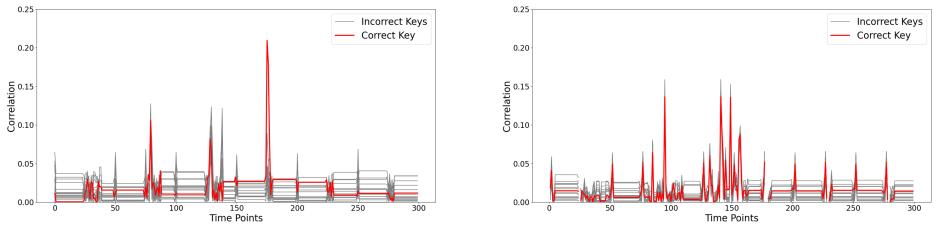


Fig. 10. Correlation traces as a function of time points obtained from unprotected (left) and protected (right) design for PRESENT.

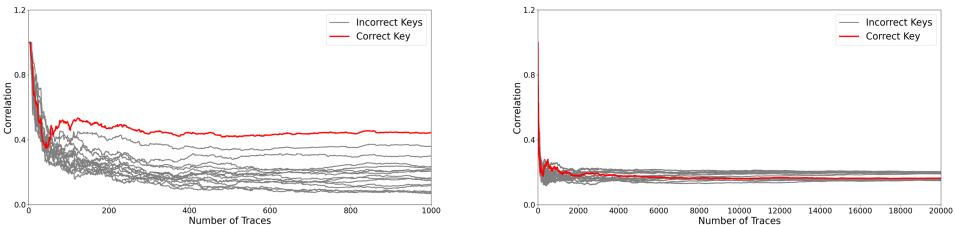


Fig. 11. Correlation traces as a function of stimuli number obtained from unprotected(left) and protected(right) design for PRESENT.

Similarly, Figure 10 demonstrates that attackers can easily obtain the accurate key from the peak of correlation traces when the PRESENT circuit is unprotected. As illustrated in Figure 11, attackers can discern the correct key using as few as 57 traces. However, after fortification, the peak of the correlation trace corresponding to the correct key is obscured by the traces of incorrect keys. As a result, attackers are unable to determine the correct key even with 20,000 traces, as shown in Figure 11. This indicates that the hardware countermeasures enhance the security of the PRESENT encryption circuit by at least a factor of 350.

To further demonstrate the effectiveness of the proposed security enhancement, the experimental results based on the T-test are presented in Figure 12 below. The figure presents the results of a T-test attack on the PRESENT circuit before and after security enhancement using 2 million traces. The results indicate that although the T-values are not entirely below 4.5 after enhancement, the overall leakage at all circuit locations has been significantly reduced. Moreover, the highest leakage risk point (around Time Points 150, which corresponds to the CPA attack location in this study, as shown in Figure 10) is no longer prominent.

Table 4 presents the overhead introduced by the proposed obfuscation method. And Table 7 enumerates the overheads associated with the path obfuscation method. Post-hardening via the

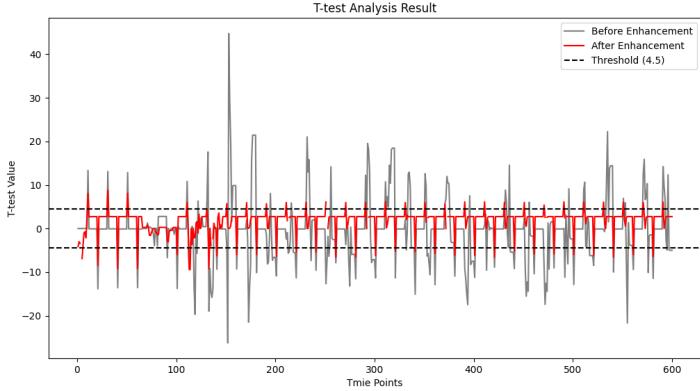


Fig. 12. T-test analysis results for PRESENT based on simulation.

Table 4. Overhead Parameters Introduced by Security Enhancement

Design	Overhead		
	Logic Cells	Power Consumption	Operating Frequency
AES-128	+659	+600 W	-1.4 MHz
PRESENT	+554	+1100 W	-1 MHz

path obfuscation method, the AES-128 necessitates 10,742 logic cells, marking an increment of 659 cells, 436 cells of which are provided by PRNGs. The PRESENT necessitates 3,216 logic cells with 554 logic cells added, 218 cells of which are provided by PRNGs. Additionally, for AES-128, the hardware countermeasures contribute an extra power overhead of 600 W, elevating the power consumption from 13.3 mW to 13.9 mW. The peak clock frequency undergoes a reduction, declining from 45.5 MHz to 44.1 MHz. For PRESENT, the hardware countermeasures contribute an extra power overhead of 1,100 W, elevating the power consumption from 6.4 mW to 7.5 mW. The peak clock frequency undergoes a reduction, declining from 53.9 MHz to 52.9 MHz.

6.3 Analysis of Actual Results

To validate the efficacy of our framework against real-world SCA attacks, we conducted empirical measurements on designs integrated into the SAKURA-G platform. To perform EM SCA evaluation, a platform is established comprising an FPGA, a probe for collecting side-channel traces, a 3D displacement stage to facilitate information gathering, an oscilloscope, and a computer to analyze the side-channel data acquired by the probe, as well as an EM shielding box that blocks external noise, which could otherwise affect the analysis results. Given that our framework is tailored for ASIC chips, it's imperative to modify the post-synthesis netlists to align with FPGA requirements. This involves redefining the foundational modules for each logic cell within the technology library. Figure 13 elucidates this process, showcasing the DFFRHQX1 logic cell as an example, detailing its functions and the corresponding rewritten RTL module. During the development phase for Xilinx FPGAs, the tools compile and instantiate post-synthesis netlists utilizing core components such as LUTs, FFs, carry-chain logic, and multiplexers. Additionally, we employ hardware primitives to replicate the functionalities of the control unit. Throughout these stages, attributes like KEEP and DONT_TOUCH are assigned to newly introduced signals and modules to inhibit unintended logic optimizations.

Functions // DFFRHQX1

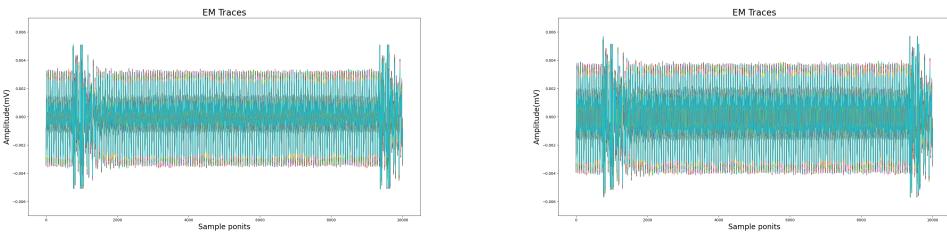
RN	D	CK	Q[n+1]
0	x	x	0
1	0	/\	0
1	1	/\	1
1	x	\/\	Q[n]

```
module DFFRHQX1 (Q, D, CK, RN);
output reg Q;
input D, CK, RN;

always@(posedge CK or negedge RN)
begin
  if (!RN)
    Q <= 1'b0;
  else
    Q <= D;
end

endmodule // DFFRHQX1
```

Fig. 13. The functions and RTL module of DFFRHQX1.



(a) Measured traces before enhancement

(b) Measured traces after enhancement

Fig. 14. Measured traces for AES-128.

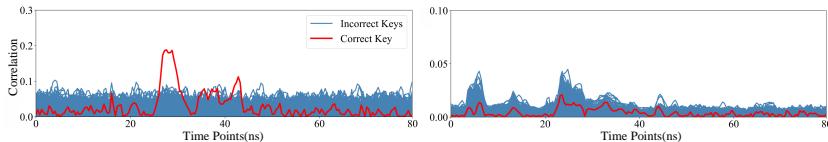


Fig. 15. Correlation traces as a function of time points obtained from unprotected (left) and protected (right) design for AES-128.

For on-chip measurements, an RF-B LANGER probe is strategically positioned near the SAKURA-G board to capture EM traces. The probe's location is meticulously adjusted until the EM trace amplitude displayed on the oscilloscope maximizes.

A static key, accompanied by 100,000 random input stimuli, is fed into the AES-128 design for encryption purposes. We amassed a total of 800,000 traces for the AES-128 design, sampled at a rate of 2.5 GSa/s. Traces collected by the probe are shown as Figure 14(a). Subsequently, we computed the mean of 8 measurements (utilizing identical input stimuli) that were temporally aligned using elastic alignment to derive the final dataset. Figures 15 and 16 present the outcomes of the CEMA attack. In the unaltered AES-128, the correlation trace of the accurate key is prominently distinguishable from those of incorrect keys. Post-framework application, the peak correlation of the correct key dwindles from 0.19 to a mere 0.02. This attenuation implies that attackers are unable to discern any information leakage. Concurrently, the CEMA attack discerns the correct key within just 92 traces for the original design (Figure 16 (left)). In stark contrast, when assailing the fortified design, the attack fails to extract the correct key even after analyzing 100,000 EM traces (Figure 16 (right)). Evidently, our tool significantly bolsters EM side-channel resistance, amplifying the MTD by at least 1085×.

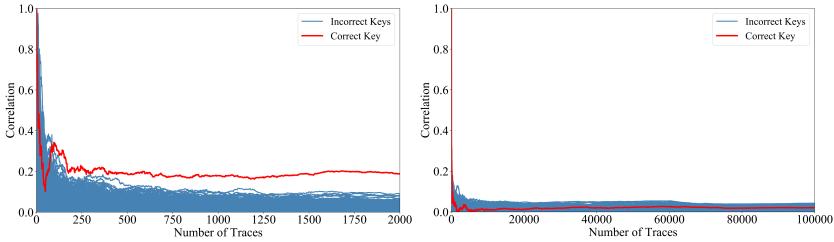


Fig. 16. Correlation traces as a function of stimuli number obtained from unprotected (left) and protected (right) design for AES-128.

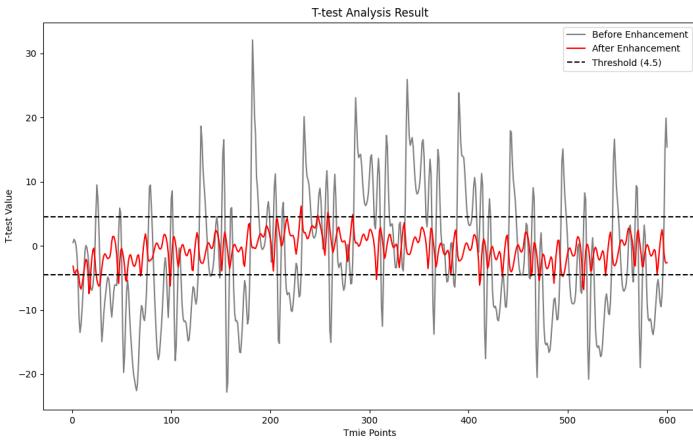


Fig. 17. T-test analysis results for PRESENT based on SAKURA-G.

To enable power-based SCA evaluation under realistic conditions, the EM acquisition probe in the aforementioned EM SCA testing platform was replaced with an SMA connector to capture power traces from the circuits executed on the SAKURA-G FPGA platform. To facilitate a comparative analysis with simulation-based evaluation results, a T-test experiment was conducted using PRESENT as the benchmark circuit.

The experimental results are presented in Figure 17. The figure illustrates the results of a T-test evaluation performed on the PRESENT circuit design implemented on the SAKURA-G FPGA platform, before and after applying the proposed countermeasures, using 2 million traces respectively. Similarly with Figure 12, the results indicate that although a few T-values still slightly exceed the threshold, the overall leakage of the circuit has been significantly mitigated. Notably, at the positions where the pre-enhancement design exhibited peak leakage, no discernible leakage is observed after applying the proposed countermeasure.

6.4 Comparison with Existing Works

Table 5 summarizes the comparison between our work and existing SCV assessment methods. Different assessment metrics are employed in the existing studies; notably, our work utilizes the Pearson Correlation Coefficient, which is the most classic evaluation metric in this field. A few studies have applied enhancement methods following vulnerability assessment. Detailed comparisons with these studies, in terms of side-channel enhancement, are shown in Table 7. For vulnerability identification performance, the comparison is depicted in Table 6 with state-of-the-art

Table 5. Comparison with Existing Vulnerability Assessment Works

Works	Application Stage	Analytical Methodology	Metric	Enhancement Methodology After Vulnerability Assessment
PLAN [3]	gate-level	Statistical	SVF^1	Feistel encryption structure
SCRIPT [4]	gate-level	Statistical	SCV^2	-
FORTIFY [5]	gate-level	Analytical	$SPCF^3$	-
ACA [8]	gate-level	Statistical	LIF^4 & PCC^7	Wave dynamic differential logic
KARNA [24]	post-placement	Statistical	$TVLA^5$ & PCC^7	Reconfigure gates
RTL-PSC [2]	RTL	Statistical	KLD^6	-
Zoni et al. [25]	gate-level	Statistical	PCC^7	Suggestions for security enhancement are given
This work	gate-level	Statistical	PCC^7	Combined obfuscation method

-Not enhanced. ¹ Side-Channel Vulnerability Factor, ² Side-Channel Vulnerability, based on Signal-to-Noise ratio, ³ Signal Probability Correlation Factor, ⁴ Leakage Impact Factor, ⁵ Test Vector Leakage Assessment, ⁶ KL Divergence, ⁷ Pearson Correlation Coefficient.

Table 6. Performance Compared with Related Work in Vulnerability Identification

Works	No.of Signals	Time consumption	Leakage path generation
FORTIFY [5]	~20,000	~8s	No
PLAN [3]	~20,000	~33hours	No
This work	~16,000	~2600s	Yes

work such as FORTIFY [5] and PLAN [3]. Our findings reveal that FORTIFY requires approximately 8 seconds to identify vulnerabilities in a benchmark containing about 20,000 signals, while PLAN takes around 33 hours. In contrast, our method takes approximately 2600 seconds for a similar task with a benchmark of about 16,000 signals. Although FORTIFY demonstrates a significant advantage in time efficiency, the time consumption of our approach is still acceptable for a statistical method. Moreover, our work supports leakage path generation, a feature not available in either FORTIFY or PLAN.

Table 7 provides a comprehensive comparison of our approach with existing methodologies, focusing on metrics such as MTD improvement, area, power, and performance overheads. Moradi et al. [6] introduced a threshold implementation for the AES design, achieving a commendable 100× enhancement in the MTD metric. However, this came at the cost of a 3× increase in area and a 2× surge in power overheads. In a subsequent effort, they refined the conventional method [26]. Yao et al. [8] addressed vulnerabilities by substituting susceptible cells with WDDL, resulting in a 4× reduction in maximum correlation and a modest 10 % increment in area. SLPSK et al. [24] championed gate-sizing, which amplified MTD by 107× without incurring overheads, albeit complicating chip fabrication processes. KF et al. [3] integrated a 4-round Feistel structure into leak-prone modules for data obfuscation. Their findings showcased a 16× boost in MTD, albeit at the expense of a 31.9 % area and a 31.25 % performance decrement. Singh et al. [11] put forth a security-centric **all-digital low-dropout (DLDO)** regulator, which bolstered power and EM SCA resilience by 4210× and 136×, respectively. However, this approach demanded a substantial area overhead of approximately 100 %, attributed to the sizable load capacitor. The technique presented in [27] achieved a 167× MTD enhancement, accompanied by 1.23× and 1.5× overheads in area and power, respectively. An advanced iteration of their methodology is elaborated upon in [12]. In contrast, for AES-128, our methodology elevates the MTD by 1190× and 1085× against power and EM SCA onslaughts, respectively. Remarkably, the incurred overheads for our hardware protection remain minimal,

Table 7. Comparison with Existing Side-Channel Enhancement Works

Works	MTD Improv.		Overheads		
	Power	EM	Area	Power	Perf.
Moradi [6]	100×	–	359 %	262 %	40 ^a
Moradi [26]	10000×	–	196 %	–	20 ^a
Yao [8]	4× ^b	–	10 %	–	–
SLPSK [24]	107×	–	0 %	0 %	0 %
KF [3]	16×	–	31.9 %	–	31.25 %
Singh [11]	4210×	136×	96.7 % ^c	32 %	10.4 %
Das [27]	–	167×	23 %	49 %	0 %
Das [12]	125000×	83333×	36.7 %	49.8 %	0 %
This Work (AES-128)	1190× ^d	1085× ^d	6.53 %	4.51 %	3.1 %
This Work (PRESENT)	350×	–	20.81 %	15.52 %	1.82 %

^aData has not been reported, ^a Increase clock cycles, ^b Decrease the maximum correlation, ^c Area overhead includes 1.9 nF load capacitor, ^d Only 100 K traces are collected limited by experiment conditions.

standing at 6.53 %, 4.51 %, and 3.1 % for area, power, and performance metrics, respectively. For PRESENT, our methodology elevates the MTD by 350× against power SCA bring overheads of 20.81 % area, 15.52 % power, and 1.82 % performance. The result shows the larger the circuit size, the more efficient our obfuscation method is.

7 Discussion and Future Work

Although the framework demonstrates excellent capabilities in formally identifying leaky paths and enhancing security, there is still room for improvement.

The obfuscation approach we discuss, particularly the random precharge method, imposes stringent sequential constraints that require users to possess advanced circuit design expertise. Additionally, leakage cells identified by the formal method not only form feedback loops within the netlist but also subject to cell-type constraints imposed by the Boolean masking scheme, posing challenges that current obfuscation techniques struggle to address effectively. To overcome these limitations, we propose considering more sophisticated methods, such as dual-rail precharge and mixed-order masking, in our future work. These approaches aim to enhance the framework's flexibility by enabling automatic analysis of loop issues and offering stronger resistance to higher-order CPA attacks.

Further, we want to clarify that our framework's effectiveness is not inherently dependent on, nor influenced by, specific technology nodes. The reason is that the genesis of circuit side-channel leakage stems from the current flow through metal interconnects. Although circuit designs across different technology nodes might differ in terms of transistor sizes and the types of logic cells utilized, the fundamental principles governing side-channel leakage remain unchanged. Therefore, our study, grounded on these principles, introduces measures for side-channel obfuscation that remain effective regardless of variations in technology nodes. On the other hand, this framework is aimed at conducting side-channel security analysis and enhancement for self-designed circuits or white-box IPs. The files pertinent to technology nodes used in this analytical and enhancement process, such as post layout netlist and technology library as shown in Figure 1, are generated during the circuit design flow. Hence, provided users maintain a comprehensive circuit design flow for a specific technology node, transitioning this framework to alternative nodes should be straightforward.

With regard to the technology library utilized in our experiments, our investigation has been confined to the 180nm CMOS logic. Currently, our team has developed a well-established circuit design flow tailored specifically for the 180nm technology node. In future work, we aim to extend our framework's compatibility to include various technology nodes, such as 55nm CMOS LL and SAED32/28nm LL, offering customizable technology libraries for users.

Additionally, the impact of different technology nodes and logic synthesis settings on the application of CPA attacks, as well as the enhancement of SMT solver performance, will be thoroughly investigated as part of our engineering implementation issues.

8 Conclusion

In this study, we introduce a groundbreaking framework tailored for automated side-channel protection. This innovative approach empowers designers to pinpoint source cells susceptible to leakage using dynamic correlation analysis. To facilitate this, we have developed a specialized tool that formally identifies these vulnerable paths. Following this identification, we employ meticulously crafted hardware strategies, including Boolean masking and random precharge, to fortify the gate-level netlist. We rigorously evaluate the efficacy of our framework through both simulations and real-world measurements on the enhanced cryptographic design. Our empirical findings confirm that our approach amplifies side-channel resistance by a factor of at least 1000 \times , all while exerting negligible effects on area, power, and performance.

References

- [1] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential power analysis. In *Proceedings of the Annual International Cryptology Conference*. Springer, 388–397.
- [2] Miao He, Jungmin Park, Adib Nahiyani, Apostol Vassilev, Yier Jin, and Mark Tehranipoor. 2019. RTL-PSC: Automated power side-channel leakage assessment at register-transfer level. In *Proceedings of the 2019 IEEE 37th VLSI Test Symposium (VTS)*. 1–6.
- [3] Muhammad Arsath KF, Vinod Ganesan, Rahul Bodduna, and Chester Rebeiro. 2020. PARAM: A microprocessor hardened for power side-channel attack resistance. In *Proceedings of the 2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* IEEE. 23–34.
- [4] Adib Nahiyani, Jungmin Park, Miao He, Yousef Iskander, Farimah Farahmandi, Domenic Forte, and Mark Tehranipoor. 2020. SCRIPT: A CAD framework for power side-channel vulnerability assessment using information flow tracking and pattern generation. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 25, 3 (2020), 1–27.
- [5] A. V. Lakshmy, Chester Rebeiro, and Swarup Bhunia. 2022. Fortify: Analytical pre-silicon side-channel characterization of digital designs. In *Proceedings of the 2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 660–665.
- [6] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. 2011. Pushing the limits: A very compact and a threshold implementation of AES. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques (ASIACRYPT)*. Springer, 69–88.
- [7] Haocheng Ma, Qizhi Zhang, Ya Gao, Jiaji He, Yiqiang Zhao, and Yier Jin. 2022. Pathfinder: Side channel protection through automatic leaky paths identification and obfuscation. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 79–84.
- [8] Yuan Yao, Tarun Kathuria, Baris Ege, and Patrick Schaumont. 2020. Architecture correlation analysis (ACA): Identifying the source of side-channel leakage at gate-level. In *Proceedings of the 2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* . IEEE, 188–196.
- [9] Danilo Sijacic, Josep Balasch, Bohan Yang, Santosh Ghosh, and Ingrid Verbauwhede. 2018. Towards efficient and automated side channel evaluations at design time. *Kalpa Publications in Computing* 7 (2018), 16–31.
- [10] Xinmu Wang, Wen Yueh, Debpriya Basu Roy, Seetharam Narasimhan, Yu Zheng, Saibal Mukhopadhyay, Debdeep Mukhopadhyay, and Swarup Bhunia. 2013. Role of power grid in side channel attack and power-grid-aware secure design. In *Proceedings of the 50th Annual ACM/IEEE Design Automation Conference (DAC)*. ACM, page 78.
- [11] Arvind Singh, Monodeep Kar, Venkata Chaitanya Krishna Chekuri, Sanu K. Mathew, Anand Rajan, Vivek De, and Saibal Mukhopadhyay. 2019. Enhanced power and electromagnetic SCA resistance of encryption engines via a security-aware integrated all-digital LDO. *IEEE Journal of Solid-State Circuits* 55, 2 (2019), 478–493.

- [12] Debayan Das, Josef Daniel, Anupam Golder, Nirmoy Modak, Shovan Maity, Baibhab Chatterjee, Donghyun Seo, Muya Chang, Avinash Varna, Harish Krishnamurthy, et al. 2020. 27.3 EM and power SCA-resilient AES-256 in 65nm CMOS through $> 350\times$ current-domain signature attenuation. In *Proceedings of the 2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 424–426.
- [13] David Knichel, Amir Moradi, Nicolai Müller, and Pascal Sasdrich. 2021. Automated generation of masked hardware. *Cryptology ePrint Archive*, (2021).
- [14] David Knichel, Pascal Sasdrich, and Amir Moradi. 2020. Silver–statistical independence and leakage verification. In *Advances in Cryptology–ASIACRYPT 2020: Proceedings of the 26th International Conference on the Theory and Application of Cryptology and Information Security, Part I*. Springer, 787–816.
- [15] Gilles Barthe, Marc Olivier Gourjon, Benjamin Grégoire, Maximilian Orlt, Clara Paglialonga, and Lars Porth. 2021. Masking in fine-grained leakage models: Construction, implementation and verification. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2, 2 (2021), 189–228. DOI : [10.46586/tches.v2021.i2.189-228](https://doi.org/10.46586/tches.v2021.i2.189-228)
- [16] Mohit Tiwari, Hassan M. G. Wassel, Bita Mazloom, Shashidhar Mysore, Frederic T. Chong, and Timothy Sherwood. 2009. Complete information flow tracking from the gates up. In *Proceedings of the 14th International Conference on ACM Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 109–120.
- [17] Wei Hu, Jason Oberg, Ali Irturk, Mohit Tiwari, Timothy Sherwood, Dejun Mu, and Ryan Kastner. 2011. Theoretical fundamentals of gate level information flow tracking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30, 8 (2011), 1128–1140.
- [18] Qizhi Zhang, Liang Liu, Yidong Yuan, Zhe Zhang, Jiaji He, Ya Gao, Yao Li, Xiaolong Guo, and Yiqiang Zhao. 2022. A gate-level information leakage detection framework of sequential circuit using z3. *Electronics* 11, 24 (2022), 4216.
- [19] Maoyuan Qin, Wei Hu, Xinmu Wang, Dejun Mu, and Baolei Mao. 2019. Theorem proof based gate level information flow tracking for hardware security verification. *Computers & Security* 85, C (2019), 225–239.
- [20] Wei Hu, Lingjuan Wu, Yu Tai, Jing Tan, and Jiliang Zhang. 2020. A unified formal model for proving security and reliability properties. In *Proceedings of the 2020 IEEE 29th Asian Test Symposium (ATS)*. IEEE, 1–6.
- [21] Jeremy Blackstone, Wei Hu, Alric Althoff, Armaithi Ardestiricham, Lu Zhang, and Ryan Kastner. 2020. A unified model for gate level propagation analysis. arXiv:2012.02791. Retrieved from <https://arxiv.org/abs/2012.02791>
- [22] Jason Oberg, Timothy Sherwood, and Ryan Kastner. 2013. Eliminating timing information flows in a mix-trusted system-on-chip. *IEEE Design & Test* 30, 2 (2013), 55–62.
- [23] Jason Oberg, Sarah Meiklejohn, Timothy Sherwood, and Ryan Kastner. 2014. Leveraging gate-level properties to identify hardware timing channels. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33, 9 (2014), 1288–1301.
- [24] Patanjali Slpsk, Prasanna Karthik Vairam, Chester Rebeiro, and V. Kamakoti. 2019. Karna: A gate-sizing based security aware EDA flow for improved power side-channel attack protection. In *Proceedings of the 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.
- [25] Davide Zoni, Alessandro Barenghi, Gerardo Pelosi, and William Fornaciari. 2018. A comprehensive side-channel information leakage analysis of an in-order RISC CPU microarchitecture. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 23, 5 (2018), 1–30.
- [26] Aein Rezaei Shahmirzadi and Amir Moradi. 2021. Re-consolidating first-order masking schemes. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021, 1 (2021), 305–342. DOI : <https://doi.org/10.46586/tches.v2021.i1.305-342>
- [27] Debayan Das, Mayukh Nath, Baibhab Chatterjee, Santosh Ghosh, and Shreyas Sen. 2019. STELLAR: A generic EM side-channel attack protection through ground-up root-cause analysis. In *Proceedings of the 2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 11–20.

Received 20 September 2024; revised 27 July 2025; accepted 30 August 2025