SCRIPT: A CAD Framework for Power Side-channel Vulnerability Assessment Using Information Flow Tracking and Pattern Generation

ADIB NAHIYAN, JUNGMIN PARK, and MIAO HE, University of Florida, USA
YOUSEF ISKANDER, Cisco, USA
FARIMAH FARAHMANDI, DOMENIC FORTE, and MARK TEHRANIPOOR, University of Florida, USA

Power side-channel attacks (SCAs) have been proven to be effective at extracting secret keys from hardware implementations of cryptographic algorithms. Ideally, the power side-channel leakage (PSCL) of hardware designs of a cryptographic algorithm should be evaluated as early as the pre-silicon stage (e.g., gate level). However, there has been little effort in developing computer-aided design (CAD) tools to accomplish this. In this article, we propose an automated CAD framework called *SCRIPT* to evaluate information leakage through side-channel analysis. SCRIPT starts by defining the underlying properties of the hardware implementation that can be exploited by side-channel attacks. It then utilizes information flow tracking (IFT) to identify registers that exhibit those properties and, therefore, leak information through the side-channel. Here, we develop an IFT-based side-channel vulnerability metric (*SCV*) that is utilized by SCRIPT for PSCL assessment. SCV is conceptually similar to the traditionally used signal-to-noise ratio (SNR) metric. However, unlike SNR, which requires thousands of traces from silicon measurements, SCRIPT utilizes formal methods to generate SCV-guided patterns/plaintexts, allowing us to derive SCV using only a few patterns (ideally as low as two) at gate level. SCV estimates PSCL vulnerability at pre-silicon stage based on the number of plaintexts required to attain a specific SCA success rate. The integration of IFT and pattern generation makes SCRIPT efficient, accurate, and generic to be applied to any hardware design. We validate the efficacy of the SCRIPT framework by demonstrating that it can effectively and accurately determine SCA success rates for different AES designs at pre-silicon stage. SCRIPT is orders of magnitude more efficient than traditional pre-silicon PSCL assessment (SNR-based), with an average evaluation time of 15 minutes; whereas, traditional PSCL assessment at pre-silicon stage would require more than a month. We also analyze the PSCL characteristic of the multiplication unit of RISC processor using SCRIPT to demonstrate SCRIPT's applicability.

CCS Concepts: • **Hardware** → *Assertion checking*; *Coverage metrics*; *Test-pattern generation and fault simulation*;

Additional Key Words and Phrases: Side-channel, vulnerability, security metric, information flow tracking, pattern generation, CAD framework

Adib Nahiyan and Jungmin Park contributed equally to this work.

This work was supported in part by Semiconductor Research Corporation (SRC), National Institute of Standards and Technology (NIST), and Cisco Systems, Inc. (Cisco)

Authors' addresses: A. Nahiyan, J. Park, M. He, D. Forte, and M. Tehranipoor, University of Florida, Gainesville, Florida; emails: adib1991@ufl.edu, jungminpark@ufl.edu, tonyhe@ufl.edu, dforte@ece.ufl.edu, tehranipoor@ece.ufl.edu; Y. Iskander, Cisco, Knoxville, Florida; email: yiskande@cisco.com; F. Farahmandi, University of Florida, Gainesville, Tennessee; email: farimah@ece.ufl.edu.

1084-4309/2020/05-ART26 $15.00
https://doi.org/10.1145/3383445

ACM Reference format:
Adib Nahiyan, 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 Trans. Des. Autom. Electron. Syst.* 25, 3, Article 26 (May 2020), 27 pages.
https://doi.org/10.1145/3383445

1 INTRODUCTION

Power side-channel attacks (SCAs) exploit the weaknesses in the hardware implementations of crypto algorithms to leak sensitive information, e.g., the encryption key. A number of SCAs such as differential power analysis (DPA) [29], correlation power analysis (CPA) [15], mutual information attack (MIA) [22], and partitioning power analysis (PPA) [30], have been proposed and successfully demonstrated over the past two decades. The underlying principle of these attacks is to exploit the relation between intermediate data and power consumption when attempting to extract the secret key [21]. Different countermeasures have also been proposed to defend against SCAs, e.g., "Masking Mechanism," which attempts to randomize the intermediate values of a cryptographic operation to mask the dependencies between these values and power consumption [10, 14, 34].

Due to the rise of power side-channel attacks, power side-channel leakage (PSCL) assessment has become very important. Without an accurate PSCL assessment, security mechanisms of the design can be useless, as attackers can bypass them by exploiting side-channel characteristics. It is meant to evaluate the amount of information leakage from crypto hardware. Different metrics, such as signal-to-noise ratio (SNR) [32, 33], t-statistic [20, 24], success rate [23], and information theoretic [43], have been proposed to analyze the vulnerability of crypto hardware to power side-channel. PSCL assessment begins with a security analyst developing an attack model followed by a verification/measurement setup. These processes involve identifying a target function, developing a power model, performing crypto operations for thousands of plaintexts, and collecting their corresponding power traces on a prototype device. The security analyst then evaluates how vulnerable the cryptographic implementation is to SCAs using the aforementioned metrics [28].

Motivations: (i) Current practices for PSCL assessment are restricted to the actual fabricated hardware, i.e., silicon (microprocessor, FPGA, or prototyped ASIC) and are not suitable for pre-silicon design and verification stages, as they require a large number of test vectors to perform PSCL assessment. While applying thousands of test vectors and collecting their power traces can be efficiently done at post-silicon, it requires prohibitively large assessment time overhead in the pre-silicon design stage using simulation. A fabricated device can operate at gigahertz (GHz) speed, which allows the verification engineer to collect the required number of power traces in seconds; whereas similar simulations could take days or weeks, depending on which level of abstraction it is performed at, e.g., physical layout level. For instance, simulation of a DPA attack on a small cryptographic circuit (the minimum stripped version of DES) at the physical layout level took 275 hours [41]. For 40 nm and lower technology nodes, even gate-level simulation may require prohibitively long simulation time for large designs and large number of test vectors [39]. (ii) Existing techniques for PSCL assessment are mostly applicable to crypto designs, e.g., AES, DES, and RSA, but are not compatible with non-crypto designs. It has been demonstrated that non-crypto designs, e.g., multiplication units of a processor, can also leak side-channel information. For example, Aysu et al. [9] presented a side-channel attack on a post-quantum key exchange protocol by exploiting the multiplication operation that depends on the subkey. Therefore, it is necessary to perform PSCL assessment of both crypto and non-crypto designs, which cannot be done by current techniques. (iii) Pre-silicon PSCL assessment has become a major interest to

semiconductor and electronic design automation (EDA) industries [44]. Identifying PSCL at early design[1] stages would save orders of magnitude in cost, while at the same time it provides utmost flexibility to address PSCL problems if they exist. (iv) Existing techniques require a team of security analysts with significant knowledge of side-channel attacks and countermeasures in each organization. It is prohibitively expensive for a semiconductor design house to maintain a large team of security experts for every vulnerability in a system-on-chip (SoC) design.

Our Contributions: Limitations of current practices for PSCL evaluation and the importance of performing this evaluation during the pre-silicon design stages suggest to automate this process and develop CAD tools for this purpose. In this article, we propose an automated framework called *SCRIPT,* for **S**ide-**C**hannel vulne**R**ability using **I**nformation flow tracking and **P**a**T**tern generation. SCRIPT can be easily integrated into the early design stages, here in gate-level, and validation stages of both application-specific integrated circuit (ASIC) and field programmable gate array (FPGA) design flows. SCRIPT utilizes an IFT-based side-channel vulnerability metric (*SCV*) for SCA vulnerability assessment. Two main engines within the SCRIPT framework are (i) information flow tracking (IFT) and (ii) SCV-guided pattern generation (SPG). IFT identifies which particular functions may contribute to PSCL and, based on this information, an SPG engine generates just a few patterns (here, plaintext[2]) that enable measuring the PSCL. The combination of these techniques makes the SCRIPT framework efficient, accurate, and generic to be applied to any hardware design (both crypto and non-crypto designs) that requires the protection of secrets or security assets against SCAs. SCRIPT can quantitatively and accurately evaluate the PSCL vulnerability of a hardware design. It also allows designers to compare PSCL resiliency of different hardware implementations of the same algorithm while considering area, delay, and power overhead. We summarize our main contributions as follows:

- We analyze various SCAs (e.g., DPA and CPA) with the aim of identifying the underlying properties of a hardware design that an attacker can exploit to perform SCAs.
- We utilize information flow tracking technique to identify registers that exhibit the properties that cause power side-channel leakage.
- We propose a metric, named SCV, for PSCL assessment. To evaluate SCV, we utilize formal methods to generate a few plaintexts (ideally as low as two plaintexts) that allow one to perform highly effective PSCL assessment. This approach, done for the first time, addresses the major limitation of previously proposed techniques that require thousands of plaintexts as input patterns for side-channel leakage evaluation and makes SCRIPT uniquely suitable for fast pre-silicon PSCL assessment.
- We present (theoretically and experimentally) a strong correlation between our proposed SCV (estimated at pre-silicon) and *SNR* metric (evaluated at post-silicon) and empirically derive the scaling factor between these two metrics. We show how the SCV metric can effectively predict the success rate (SR) of power side-channel attacks given n plaintexts.
- We perform case studies on Galois field and look-up table–based AES hardware implementations on Xilinx FPGAs. Our experimental results demonstrate that SCRIPT can effectively assess PSCL vulnerability at pre-silicon as validated with silicon results.
- Finally, we demonstrate that SCRIPT can be applied to any hardware design (e.g., AES, DES, and even non-crypto modules) without the need for customization and without significant security expertise by analyzing the PSCL of the multiplication unit of RISC processor using SCRIPT.

[1] "Hardware design" and "design" are used interchangeably throughout this article, referring to gate-level netlist of the design.

[2] Patterns and plaintexts are used interchangeably throughout the article.

The rest of the article is organized as follows: Section 2 briefly discusses the prior work on side-channel leakage assessment. Section 3 presents the SCRIPT framework in detail. Section 4 presents our results. Section 5 concludes the article. Appendices show the mathematical proof of our proposed theorems.

2 PRIOR WORK

Messerges et al. [33] proposed the signal-to-noise ratio metric (SNR) to evaluate the effectiveness of DPA attacks where a higher SNR value indicates a more effective attack. Gierlichs et al. [22] proposed a distinguisher based on mutual information between the observed power traces and hypothetical power leakage to rank key guesses. Fei et al. [21] proposed a general statistic model based on maximum likelihood estimation that can model and estimate the success rate (SR) of side-channel attacks such as DPA or CPA. However, these techniques are designed to assess side-channel leakage at the post-silicon evaluation stage and are not suitable for the pre-silicon stage, as they require many plaintexts and/or measurement traces.

Gilbert et al. [24] proposed to use TVLA test to evaluate the side-channel resistance of a cryptographic module. In this technique, the collected power traces are partitioned into two sets based on the intermediate values related to the secret information. Then the t-statistic is calculated to quantify the confidence level of the two sets being statistically different. A t-threshold value of 4.5 was used to test if the device leaks a side-channel information. However, this technique only provides a pass/fail (not the amount of leakage) test and may produce false positive results [35]. Moradi et al. [35] propose to use Pearson's χ^2-test and t-test to address some limitations of TVLA. However, similar to previous approaches, these techniques are also not suitable for pre-silicon side-channel vulnerability assessment, as they require many plaintexts/traces.

Veshchikov et al. [42] presented a comprehensive survey of simulators for side-channel analysis, e.g., PINPAS [18], SCARD [8], NCSim [19], OSCAR [40], and so on. Among these reported side-channel simulators, only two (SCARD and NCSim) work with hardware crypto modules, whereas the rest deal with software crypto algorithms implemented in microprocessors. These simulators, in general, emulate the execution of a program under analysis and use simple power estimate techniques, e.g, Hamming weight or Hamming distance. In addition to not being applicable to ASICs and FPGAs, these simulators cannot produce accurate results, as their power estimation model is not derived from the actual physical characteristic of the implemented design. SCARD, however, aims to develop an accurate spice model for power side-channel leakage simulation. However, SCARD requires simulation at spice level for many input patterns, which is not feasible as well as cannot be used in pre-silicon when the prototype device is not available. NCSim utilizes commercial power estimation tool to evaluate DPA resistance at the gate level. However, this technique also requires thousands of plaintexts and, therefore, may require prohibitively large assessment time overhead. Bayrak et al. [11–13] propose automated application and verification of power analysis countermeasures on software implementation of crypto algorithms. However, these techniques cannot be applied to hardware PSCL.

Huss et al. [27, 28] proposed a framework named AMASIVE for side-channel vulnerability assessment. AMASIVE identifies the hypothesis function for Hamming Weight/Hamming Distance model, which is used for side-channel vulnerability assessment. However, this framework can only identify the hypothesis function, and the final vulnerability assessment still needs to be performed on a prototype device requiring thousands of plaintexts. Therefore, this technique also is not suitable for pre-silicon PSCL evaluation.

We summarize the assessment time and accuracy of PSCL assessment as well as the flexibility to make design changes at different pre-silicon design stages w.r.t. the post-fabricated device level in Table 1. PSCL assessment at different design stages has a trade-off between time/accuracy and

Table 1. Comparison: Pre- and Post-silicon PSCL Evaluation

	Pre-silicon			Post-silicon
	RTL	Gate-level	Layout	
Time	Medium	High	Very High	Low
Accuracy	Low	Medium	High	Very High
Flexibility	High	Medium	Low	Not Feasible (ASIC); Difficult (FPGA)

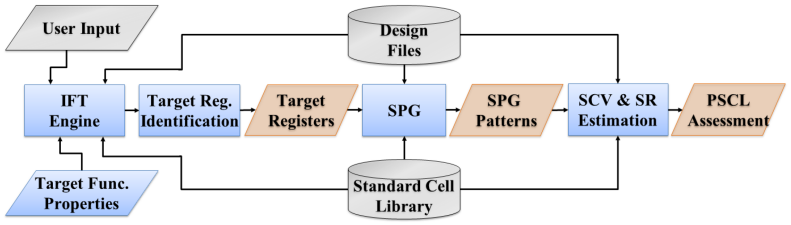

Fig. 1. Overview of the SCRIPT framework for power side-channel leakage (PSCL) assessment. PSCL assessment refers to estimating SCA success rate at pre-silicon stage and predicting the number of plaintexts for a successful SCA.

flexibility. While the post-silicon PSCL assessment has the highest accuracy and the fastest processing time, flexibility to make design changes to address potential vulnerabilities is the worst. For ASIC, any post-fabrication modification is not feasible; whereas, for FPGA post-synthesis, bitstream modification is difficult. However, pre-silicon PSCL evaluation offers better flexibility to address potential vulnerabilites at the cost of poor accuracy and long processing time. In this article, we propose an accurate and efficient pre-silicon PSCL assessment technique to address the limitations of existing approaches.

3 PROPOSED CAD FRAMEWORK: SCRIPT

In this section, we describe the SCRIPT framework for PSCL assessment at the gate level. A high-level overview of the SCRIPT is shown in Figure 1. We first extract the underlying properties of the function that cause side-channel leakage. We then utilize information flow tracking (IFT) engine to identify registers (termed as target registers) that exhibit these properties. Our IFT engine works on the gate-level netlist and takes some inputs from the user, e.g., the name of the key and plaintext input ports. We then employ formal verification technique to generate specific input patterns (plaintexts) that produce maximum power difference in the target registers. The difference in the power consumption of the target registers can be exploited by the adversary during the side-channel attack. We then evaluate our proposed SCV metric to provide a quantitative assessment of how vulnerable the design is to side-channel attack. Before we provide details of the SCRIPT framework, we first present the threat model employed in this framework.

3.1 Threat Model

SCRIPT framework is designed to be used by the semiconductor design house that develops the hardware crypto accelerator for the purpose of identifying potential side-channel leakage vulnerability. The verification engineer using SCRIPT has access to the gate-level design of the crypto accelerator along with the standard cell libraries. The verification engineer has the white-box knowledge of the design, i.e., knows the ports corresponding to the key, plaintext, and ciphertext; however, he/she does not have the skill set for performing SCA or analyzing vulnerabilities to SCA. Therefore, he/she will use the SCRIPT framework for performing PSCL assessment.

From a side-channel vulnerability assessment point of view, our framework intends to find the worst-case leakage scenario. Therefore, we consider a strong attack model where the attacker performing the side-channel attack has in-depth knowledge of the cryptographic algorithm and has some knowledge about the implementation. For example, the attacker knows at which time instance a specific round operation occurs. We also assume that the attacker has full control over the plaintext and can perform any number of encryption operations on plaintext data. The attacker also has physical access to the power port, which allows him to observe the power traces. Note that in this work, we consider that the attacker uses the plaintext input to perform side-channel attacks for simplicity. However, SCRIPT can also be easily extended to the case where the attacker can observe the ciphertext output to perform SCA attacks.

3.2 Properties of Target Function

A function needs to have specific properties to be targeted for side-channel attacks. We have reviewed existing literature on DPA and CPA as well as template and profiling attacks on hardware implementations of different symmetric encryption algorithms, e.g., AES, DES, and PRESENT. Based on our review, we have extracted the underlying properties of a function that are inherently exploited by side-channel attacks. We define these properties as follows:

- **P1: Function of the secret.** The target function should be a function of the secret information, e.g., encryption/decryption key or an intermediate value that is related to the key. As a simple example, the AES SBox operation is a function of the key.
- **P2: Function of the controllable inputs.** The target function should also be a function of a variable or an input, e.g., plaintext, that an attacker can control. This property allows an attacker to control the input of the target function to create the power hypothesis model and guess the intermediate values of the secret. For example, the AES SBox operation of the first round exhibits this property, as it is the function of the plaintext that an attacker can control.
- **P3: Function with confusion property.** This property dictates that the target function needs to possess the confusion property, i.e., one output bit of the target function should depend on more than one key input bit. Confusion property is quantified by the confusion coefficient metric [31], which measures the probability that, given two different key guesses, the respective power hypothesis model is different. This property ensures that an adversary can isolate the correct key hypothesis from the guessed key space. The AES SBox exhibits this property with high confusion coefficient metric.
- **P4: Function with divide-and-conquer property.** The target function also needs to be a function of a small subset of the secret information. This property allows the adversary to apply divide-and-conquer strategy to focus on one subset of the secret information at a time and extract the subset. The AES SBox operation also has this property, since it depends on the eight-bit subkey.

A function having these properties is defined as "Target Function" and denoted by T. Note that, as an example of the target function, we have used the AES SBox operation. However, other crypto functions (e.g., AddRoundKey of AES) or non-crypto functions (e.g., logic obfuscation [45]), having the aforementioned properties, can also be used as target functions. We formally define the target function as follows:

$$T = \{f(k', p') |\ k' \subset k\ \&\ size(k') > 1\}, \tag{1}$$

where k is the key and p' is the variable that an attacker can control. $k' \subset k$ represents the divide-and-conquer property, whereas, $size(k') > 1$ represents the confusion property. These four

properties are prerequisites for DPA and CPA as well as template and profiling SCA attacks. However, not all side-channel attacks exploit all four of these properties. For example, simple power analysis attack does not have controllable input requirement (property 2). Therefore, these four properties do not guarantee that they are sufficient for all possible side-channel attacks. However, existing properties could be modified or additional properties can be added to cover such attacks as well.

Target Register: We define the registers that store the output values of the target function (i.e., T) as target registers. The switching characteristic of the target registers embeds the four properties of T that are exploited by side-channel attacks. Therefore, the switching power of these registers and their corresponding fan-in combinational logic gates contribute greatly to power side-channel leakage. The SCRIPT framework autonomously identifies target registers using information flow tracking, which is described in the following subsection.

3.3 Identifying Target Registers Using IFT

We propose to utilize information flow tracking to identify target registers that meet the properties discussed in Subsection 3.2. There exist multiple techniques for tracking information flow in hardware. For example, Hu et al. [26] proposed the gate-level information flow tracking (GLIFT) technique where each gate in the design is augmented with taint shadow logic for tracking taint propagation. GLIFT is a powerful technique that can visualize the propagation of taint as well as the logic value for each net. However, GLIFT in general is a simulation-based approach where a testbench is required for inserting taint values at the source of the paths and checking whether the destination signals can receive the taint value. This method is not complete, because it can show the information flow path illustrated in the testbench, but it cannot prove their absence [25].

For SCRIPT, we utilize the concept of employing fault propagation to track information propagation [17, 36]. Our IFT technique does not require any testbench and automatically proves the existence of information flow key carrying net to target registers. In addition, our IFT automatically derives the input vectors that enable this information propagation. We utilize the stimulus vector to check if the registers show the properties of the target function. Our IFT engine can be applied to the gate-level netlist of a design. It utilizes existing automatic test pattern generation (ATPG) tools[3] such as Tetramax from Synopsys, Fastscan from Mentor Graphics, or Encounter Test from Cadence, which are commercially available and widely used by industry. In this work, we have used Tetramax [5] along with our developed Tcl script to implement the IFT engine.

Below, we first briefly describe our IFT engine and then discuss how we utilize IFT to satisfy function property P1 (P1: Function of the secret) followed by identification of target registers.

3.3.1 IFT Engine. Our IFT engine is based on the concept of modeling a secret information (e.g., the encryption key) as a stuck-at-0 and stuck-at-1 fault[4] and leveraging the ATPG tool [16] to detect those faults. A successful detection of faults means that the logical value of the key carrying net can be observed through the observe points, i.e., the registers. In other words, there exists an information flow from the fault location to the observe points. ATPG employs a path sensitization algorithm [16] that exhaustively analyzes if there exists any information flow path from a key bit to an observe point and automatically derives the input vectors that enable this information propagation.

Our IFT engine needs to identify all registers (or flip-flops) where a key bit propagates to. However, traditional full-scan and full-sequential ATPG analysis cannot be used for this purpose [36].

[3]ATPG is an electronic design automation technology that can derive test patterns in an automated fashion that can distinguish between correct and faulty circuit behavior [16].
[4]Stuck-at-fault is modeled by assigning a fixed logic value (0 or 1) to a single net or port of a design [16].

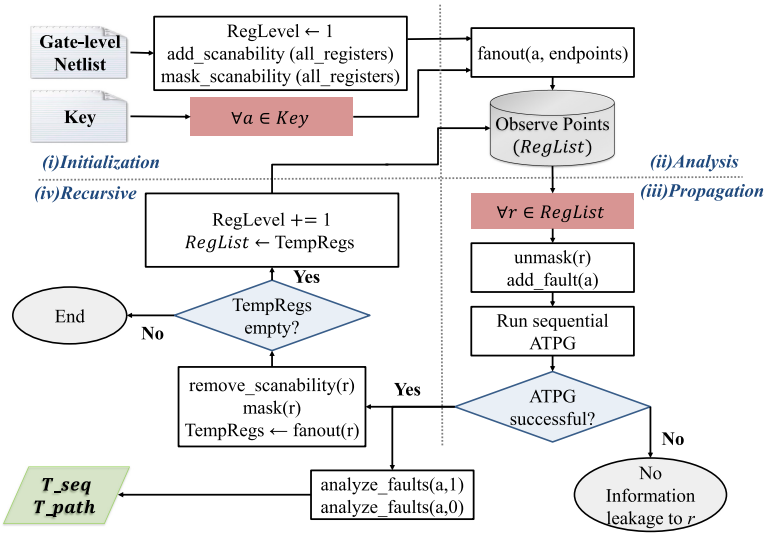

Fig. 2. IFT algorithm utilizing partial-scan ATPG to identify the registers where a key bit propagates to.

The full-scan ATPG can only identify the first-level (sequential depth = 1) registers[5], whereas, the full-sequential ATPG is inefficient.[6] We address this limitation by utilizing *partial-scan ATPG* technique. In a partial-scan design, the scan chains contain some, but not all, of the sequential elements in the design. Traditionally, partial scan is used to minimize area overhead (caused by design for test structure) while attaining the targeted test coverage. However, our partial-scan approach for information flow tracking is fundamentally different from the traditional one. In our IFT technique, we use the partial-scan ATPG to identify the observe points through which a key bit can be observed. Note that the partial-scan ATPG will only be used for our IFT engine. Once the verification is complete, the design can be transformed back into its original form. Figure 2 shows the overall flow of our IFT engine and its four main steps: Initialization, Analysis, Propagation, and Recursive.

(i) Initialization: This step takes the name of the key input ports to which IFT will be applied (shown in Figure 2 as *key*), the gate-level netlist of the design, and the technology library (required for ATPG analysis) as inputs. Then, the engine adds scan capability to all the registers/flip-flops (FFs) in the design to make them controllable and observable. Here, we use the "What If" [5] analysis feature to virtually add and/or remove FFs from scan chain. This feature allows us to perform partial-scan analysis dynamically without requiring to re-synthesize the netlist. We also apply masks to all FFs so we can track key propagation to each FF independently. Applying masks is an important step, as it allows controlling fault propagation to one FF at a time.

(ii) Analysis: This step utilizes fanout analysis for each key bit $a \in key$ to find the FFs that are in the fanout cone of a. These FFs are potential candidates of target registers, as a key bit a can

[5]Full-scan ATPG converts all sequential elements, i.e., flip-flops (FFs), in the design to scan FFs where each scanned FF can act as a control or observe point. Although being efficient, full-scan ATPG only allows fault propagation to first-level registers. Fault propagation to the subsequent level of registers is blocked by the first-level scan registers [17].

[6]In full-sequential ATPG, no sequential elements, i.e., flip-flops (FFs), are converted to its scan equivalent and, therefore, none can act as a control or observe point. Full-sequential ATPG needs to search through the entire functional input space to propagate a fault to an observe point, which has proven to be ineffective and extremely time-consuming, hence, typically not used [17].

potentially propagate to them. This step generates a list of potential candidates of target registers (called *RegList*). This list is used in the next stage, as shown in Figure 2.

(iii) Propagation: This step tracks the propagation of each key bit a to each individual FF. To perform a comprehensive analysis of potential points of the key bit propagation, each FF must be analyzed separately. For each $r \in RegList$ (shown in Figure 2), the applied mask is removed, so the key bit propagation to r can be tracked. The next step adds the key bit a as the only stuck-at fault in the design and runs ATPG algorithm in the sequential mode to find paths to propagate $a = 0$ and $a = 1$ to FF r. If both $a = 0$ and $a = 1$ can be detected from r, then there exists an information flow from a to r and r is marked as a potential target register. This step also stores the propagation path (T_{path}) as well as the control sequence (T_{seq}) required for the key bit propagation for further analysis. Note that T_{seq} contains the list of input ports and control registers that controls the information propagation from a to r.

(iv) Recursive: This step leverages our partial-scan technique along with sequential ATPG to find propagation paths through all sequential levels until the output or the last-level FFs are reached. Here, the function *remove_scanability* (shown in Figure 2) makes the ATPG tool treat r as a non-scan FF for simulation purposes without redoing scan chain insertion. The FF's output ports Q and QN are used to get a new fanout emerging from r to the next level of registers. To find information flow through multiple levels of registers, the scannability of all identified registers in *RegList* is removed incrementally and sequential ATPG is used to create propagation paths from key bit a to subsequent-level registers. This process continues until the last level of registers.

The output of our IFT engine is a list of registers/FFs where the key propagates to, i.e., key observe points (*RegList*), and the propagation path (T_{path}) along with the stimulus vector (T_{seq}) for asset propagation for each FF, r. In the following subsection, we discuss how we utilize this information to identify target registers.

3.3.2 Target Registers Identification. Once we have identified the registers $RegList_a$ where a key bit a propagates to, we analyze the stimulus vector (T_{seq_a}) to check if the registers show the properties of the target function (discussed in Section 3.2). Note that all registers in $RegList_a$ satisfy the first property of *P1: Function of secret information*. The reason is that the key bit a propagates to $RegList_a$. We utilize Algorithm 1 to analyze which registers in $RegList_a$ satisfy the remaining three properties (P2–P4).

Algorithm 1 first takes the *RegList* and T_{path} as inputs (Line 2). Then for each r in the *RegList*, Algorithm 1 extracts the sequential depth T_SD_r from T_{path} (Line 5). Next, the algorithm performs the successive fanin analysis to find the (i) control key port names T_CK_r and (ii) control plaintext port names T_CPI_r (Lines 6–7). Here, $FanIn(r, T\_SD_r)$ refers to performing fanin analysis of r up to the sequential depth T_SD_r. Algorithm 1 then checks for the following properties:

- Whether one or more plaintext input ports control key bit propagation to r ($length(T\_CPI_r) \geq 1$). If yes, then property *P2: Function of the control input* is satisfied.
- Whether more than one key input ports control key bit propagation to r ($length(T\_CK_r) > 1$). If yes, then property *P3: Function with confusion property* is satisfied.
- Whether the number of key input ports that control key bit propagation to r is less than K_{Th} ($length(T\_CK_r) < K_{Th}$). If yes, then property *P4: Function with divide-and-conquer property* is satisfied. Here, we set the K_{Th} value to be 32. K_{Th} determines the number of possible key guesses ($2^{K_{Th}}$) for SCAs. Note that $K_{Th} = 32$ does not refer to the key length of an encryption algorithm but refers to the maximum limit considered for the divide-and-conquer property.

If all properties are satisfied (Line 8), then Algorithm 1 marks r as a target register (Line 10) and stores $\{r, T\_CK_r, T\_CPI_r, T\_SD_r\}$ in a G_r variable (Line 11). After completing the above analysis

ALGORITHM 1: Target Registers Identification

```
1:  procedure Identifying and Grouping Target Registers
2:  Input: RegList, T_path
3:  Output: Group of Target Registers
4:      for all r ∈ RegList do
5:          T_SD_r ← Extract sequential depth from T_path
6:          T_CK_r ← FanIn(r, T_SD_r) ∈ Key
7:          T_CPI_r ← FanIn(r, T_SD_r) ∈ Plaintext
8:          if ((length(T_CPI_r) ≥ 1) & (length(T_CK_r) > 1)&
9:            (length(T_CK_r) < K_Th)) then
10:             r → Target Register
11:             G_r ← {r, T_CK_r, T_CPI_r, T_SD_r}
12:         end if
13:     end for
14:     for <i = 1, i < length(G_r), i++> do
15:         for <j = i, j < length(G_r), j++> do
16:             if G_r(i) : T_CK_r == G_r(j) : T_CK_r then
17:                 G_r(i) = {G_r(i)||G_r(j)}
18:             end if
19:         end for
20:         end for
21:     end procedure
```

for all $r \in RegList$, the algorithm analyzes the G_r variables to place all r with the same control key bits (T_CK_r) in the same group (Lines 14–18). It is an important step, as it identifies which key bits control which particular set of target registers.

Note that Algorithm 1 identifies registers that satisfy all four properties of the target function. These properties are prerequisites for DPA and CPA as well as template and profiling attacks. However, some power side-channel attacks do not meet all of these four properties. For example, simple power analysis attack (SPA) does not have the P2: Function of the controllable inputs requirement. We can relax the $length(T\_CPI_r) \geq 1$ constraint in Algorithm 1 (Line 8) so that our IFT can search for target registers of SPA. Similarly, new properties, i.e., constraints, can be added to cover future SCA attack.

3.3.3 Target Registers of AES. In this subsection, we utilize a Galois field-based AES design as an example to illustrate how SCRIPT utilizes the IFT engine along with the Algorithm 1 to identify target registers of this AES implementation. The details of the AES design are discussed in Section 4.1.

We first start our analysis from a key bit 32 ($KEY[32]$). Note that any key bit can be analyzed and IFT engine will find the corresponding target registers, if they exist. Figure 3 illustrates the target register identification of the AES implementation. IFT engine first searches for the registers where the $KEY[32]$ bit propagates to. It finds the data register $Data\_Reg[32]$ (which stores the intermediate round results) along with some key registers (which stores the intermediate key expansion results). Then SCRIPT framework checks whether these registers satisfy the properties of the target function (discussed in Section 3.2) using the target register identification algorithm (discussed in Algorithm 1). SCRIPT finds that the key registers are not controlled by plaintext inputs and, therefore, property *P2: Function of the control input* is not satisfied. For $Data\_Reg[32]$, SCRIPT finds that only a key bit controls this register and, therefore, *P3: Function with confusion property* is not satisfied. Therefore, the register is not a target register and is marked as red in Figure 3. Then, the SCRIPT framework searches for the second level registers. Now, it finds $Data\_Reg[0\ to\ 31]$ along with some key registers. SCRIPT finds again that key registers do not possess *P2* property

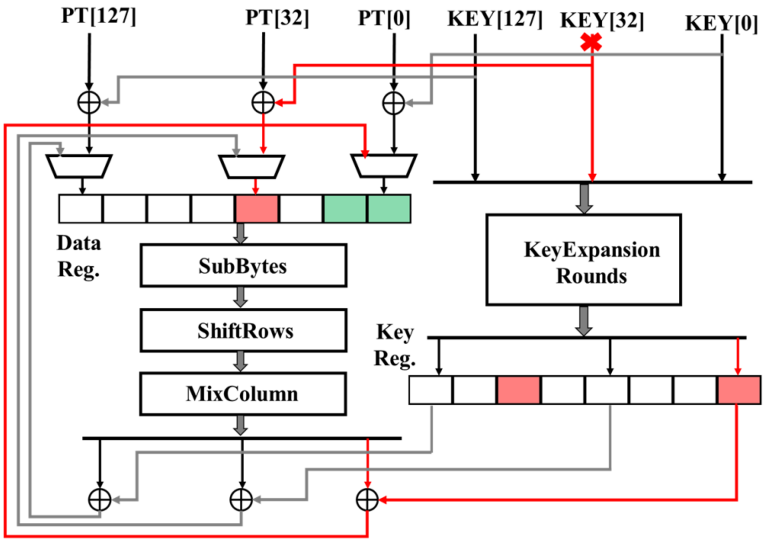

Fig. 3. Hardware architecture of the Galois field–based AES design. The red marked lines illustrate the IFT process. The green-marked registers represent Target Registers.

and, therefore, are not target registers. The *Data_Reg*[0 *to* 31], however, are controlled by 32 key inputs and 32 plaintext inputs and, therefore, *Data_Reg*[0 *to* 31] possess all the properties of Algorithm 1 and are the target registers (marked in green in Figure 3). SCRIPT continues to search for subsequent level registers. However, all the *Data_Reg* after the second level violate property *P4: Function with divide-and-conquer property* and, therefore, are not target registers. At the end, SCRIPT returns the *Data_Reg*[0 *to* 31] along with the 32 controlling key and plaintext input port names and the sequential depth 2.

Next, we analyze if in fact the *Data_Reg*[0 *to* 31] registers are the actual target registers. Our analysis shows that at the second clock cycle (derived from IFT reported sequential depth 2), these registers store the first round intermediate values, i.e., the outputs of the "SBox," "ShiftRow," and "MixColumn" operation that are actually the target for SCAs.

Here, one could argue that the target registers in a design may be known to the verification/security engineers, and, therefore, there is little need in using such identification technique. The counter-argument is that SCRIPT is a very generic CAD tool for PSCL assessment of any design including non-crypto modules for asset leakage assessment. In Section 4.6, using SCRIPT, we perform PSCL assessment of a multiplication unit of RISC processor. While a verification engineer may sometimes know which registers are responsible for PSCL for well-known crypto modules, e.g., AES, he/she may not have this knowledge for non-crypto designs, e.g., a multiplication module. Hence, this is an important step that extends SCRIPT's capability to analyze crypto as well as non-crypto designs for PSCL. The following section discusses how the identified target registers are utilized for PSCL assessment.

3.4 SCV Metric

We propose an IFT-based side-channel vulnerability metric (*SCV*) for PSCL assessment at the pre-silicon design stage. *SCV* is defined as follows:

$$SCV = \frac{P_{signal}}{P_{noise}} = \frac{P_{T.hi} - P_{T.hj}}{P_{noise}}, \tag{2}$$

where the P_{signal} refers to the difference in power consumption during the target function operation and P_{noise} refers to the the power consumption of the rest of the design. P_T represents the average power consumed when performing the target function. Let us consider that the target function consumes $P_{T.hi}$ and $P_{T.hj}$ power when the Hamming weight (HW) of the output of the target function is $hi = HW(T_i)$ and $hj = HW(T_j)$, respectively, for ith and jth inputs. The difference between $P_{T.hi}$ and $P_{T.hj}$ is exploited during the SCA. Therefore, $P_{signal} = P_{T.hi} - P_{T.hj}$. Note that we utilize the HW model [33] for SCV, which assumes that the amount of information leakage through the power side-channel depends on the number of bits set to logic 1 in the target function.

Note that the definition of our SCV metric is similar to the signal-to-noise ratio (SNR) metric [33], which has been used to evaluate the side-channel leakage assessment at post-silicon stage. The SNR metric employs the formula SNR $= \frac{\epsilon^2}{\sigma_N^2}$, where ϵ refers to the difference of mean value obtained during the DPA attack and σ_N refers to the variance of power signals. Our SCV utilizes an alternative formula for side-channel leakage assessment metric. We will show in our experimental results that the SCV measured during the pre-silicon design stage has a high correlation to the experimentally calculated SNR at post-silicon. The SCV metric only differs from SNR by a scaling factor. We establish the mathematical relationship between our estimated SCV metric and the SNR metric in Appendix B and empirically derive the scaling factor.

Note that SNR calculation requires performing the DPA attack on the target prototype device, which typically needs thousands of plaintexts [33]. While this approach for estimating SNR may be feasible at post-silicon, it is not possible at the pre-silicon stage due to slow simulation time. To address this major limitation, SCRIPT utilizes a formal verification technique to derive directed/guided patterns (ideally two) as well as utilize vector-less power estimation technique for SCV calculation. Sections 3.4.1 and 3.4.3 discuss derivation of P_{signal} and P_{noise}, respectively.

3.4.1 SPG (SCV-guided Pattern Generation). We propose SPG, a novel approach to estimate P_{signal} using a formal verification technique. Here, we utilize formal verification to generate a few specific patterns (plaintexts in case of cryptographic algorithms) and use those in functional simulation tool along with power estimation tool to get power corresponding to the patterns. Here, SCV-guided patterns refer to patterns that have the following two properties:

- The SCV-guided patterns need to produce our desired Hamming weight at the target registers.
- The SCV-guided patterns should only induce switching in the logic related to the target function while muting the switching of the rest of the design.

These properties allow us to estimate the power consumption only for producing our desired HW at the target function. In other words, we can utilize SPG to estimate the $P_{T.hi}$ and $P_{T.hj}$ values (discussed earlier in this subsection). The SPG technique generates the constraints along with necessary assertions and utilizes a formal verification technique to derive the SCV-guided patterns. The details of the SPG algorithm are presented in Algorithm 2.

Algorithm 2 first takes the list of target registers (r), control plaintext input ports (T_CPI), and sequential depth (T_SD_r) as inputs from the IFT engine (discussed in Section 3.3.1). It also takes the desired Hamming weight $hi = HW(r)$ and the design files from the user. Then, Algorithm 2 creates some constraints for the formal verification tool. It first applies a fixed input constraint for all key bits and plaintext bits that are not among T_CPI (Lines 5–6). We apply logic 0 for all fixed input constraints. Then, the algorithm applies a bind constraint to T_CPI (Line 7), which tells the tool to use the same input plaintext bits in T_CPI for all clock cycles. These constraints ensure that switching only occurs in the logic of the target function while muting the switching of the rest

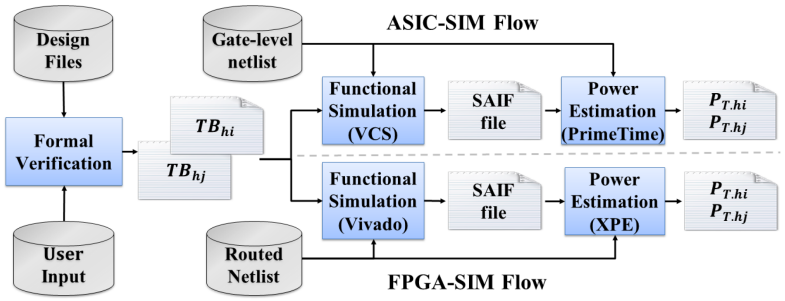

Fig. 4. Overall flow for estimating power using SPG. The ASIC-SIM flow shows the gate-level power simulation commonly used for ASIC chips. The FPGA-SIM flow shows the power estimation for Xilinx FPGAs.

ALGORITHM 2: SPG Algorithm

1: **procedure** Formal Verification based Pattern Generation
2: **Input:** Design files, $G_r\{r,\ T\_CPI_r,\ T\_SD_r\}$, hi
3: **Output:** SCV-guided Pattern (PI_{hi})
4: **#Constraints**
5: $Apply\ fixed\ input\ constraint \rightarrow K$
6: $Apply\ fixed\ input\ constraint \rightarrow PI \notin T\_CPI_r$
7: $Apply\ bind\ input\ constraint \rightarrow T\_CPI_r$
8: $Apply\ Reset$
9: Run
10: $Remove\ Reset$
11: **#Assertion**
12: $psl\ A1:\ assert\ never\ ((HW(r) == hi)\ \&\&\ (clk\_count == T\_SD_r))$
13: **#Testbench Generation**
14: $Prove$
15: $Export\ testbench \rightarrow PI_{hi}$
16: **end procedure**

of the design. We also apply "Reset" constraints to ensure that every pattern generation process starts from the same initial condition (Lines 8–10).

Next, Algorithm 2 develops the assertions that express the desired behavior of a design under test. In our case, the assertion represents the property when the Hamming weight of the r is hi and when the clock count is equal to the sequential depth (T_SD_r) (Line 12). The latter part is important, because it tells the formal tool to prove the assertion for clock cycles when the key propagates to the "target register." Then, we negate the assertion (with *never* statement) and run the formal verification tool to generate a counter example for the given assertion and generate a sequence of input patterns that causes the negation of the assertion to fail. In other words, by forcing the formal tool to verify the negation of the assertion, the tool generates patterns that satisfy this condition: $(HW(r) == hi)\ \&\&\ (clk\_count == T\_SD_r)$. Last, we export the patterns in Verilog testbench format that contain the SCV-guided patterns (PI_{hi}). In this article, we have used the Incisive Formal Verifier [2] tool from Cadence for performing the formal verification.

Once we have derived the patterns and testbenches TB_{hi} and TB_{hj} for hi and hj, respectively, we need to estimate the power difference corresponding to these testbenches. Figure 4 shows the overall flow for estimating power for the SCV-guided patterns. It can be categorized into ASIC and FPGA flow. Here, $-SIM$ in the figure refers to power estimation for ASIC and FPGA flow in pre-silicon level via simulation. Note that both are identical except for the simulation tools used. For both ASIC and FPGA, we first use a functional simulation tool to simulate TB_{hi} and TB_{hj}. Here,

we use a specific command ($\$set\_toggle\_region$) to generate the switching activity file (.SAIF file) for each testbench. Note that we set the toggle region only for the clock cycle when the target function is running. We extract this information from T_SD_r. For ASIC design flow, we use the VCS tool from Synopsys [5] and in case of FPGA, we use Vivado from Xilinx [7]. For our later experiments, we use the gate-level design for ASIC design flow and we use the place and route design for FPGA flow. Note that our framework is also compatible with routed designs. Once we have generated the .SAIF file, we feed this file to a power estimation tool to get the power $P_{T.hi}$ and $P_{T.hj}$ corresponding to hi and hj, respectively. We use the PrimeTime tool from Synopsys and XPE tool from Xilinx for power estimation. Last, we derive $P_{signal} = P_{T.hi} - P_{T.hj}$ for calculating the SCV metric.

Note that the SPG helps to isolate the difference in power consumption during the target function execution and estimate P_{signal} using just two patterns/plaintexts. We can also generate SCV-guided patterns that produce greater P_{signal} by choosing greater difference between Hamming weight hi and hj. This enables estimation of the SCV metric for a worst-case scenario from the PSCL perspective, which in our opinion should be the ideal configuration for PSCL assessment. In the experimental result section, we will show that SPG can create greater P_{signal} as compared to random patterns/plaintext (see Section 4.3.1). Also note that the difference in power consumption for both target registers and their corresponding fanin combinational logic is exploited by SCA. Our proposed SPG technique considers the power consumption caused by the switching activity of both the target registers and the combinational gates located in the target registers' fanin cone.

3.4.2 SPG for AES. In this subsection, we utilize the Galois field–based AES design (discussed in Section 3.3.3) as an example to illustrate how SCRIPT employs Algorithm 2 to generate SCV-guided patterns. We first take the list of target registers ($r = Data\_Reg[0\ to\ 31]$), control plaintext input ports ($T\_CPI = PT[0\ to\ 31]$), and sequential depth ($T\_SD_r = 2$) as inputs from the IFT engine (see Section 3.3.3). Then, we set our desired Hamming weight to $hi = 32$ to produce the worst-case power side-channel leakage for AES. Next, we apply logic 0 for all fixed input constraints to all key ports ($KEY[0\ to\ 127]$) and plaintext ports ($PT[32\ to\ 127]$), which are not among T_CPI, as well as apply bind constraint to T_CPI ($PT[0\ to\ 31]$). These constraints ensure that switching only occurs in the target function of AES while muting the switching of the rest of the AES design. Next, we write the assertion, $assert\ never\ ((HW(r) == hi)\ \&\&\ (clk\_count == T\_SD_r))$, where the Hamming weight of the r is hi and when the clock count is equal to the sequential depth T_SD_r. Then, we run the formal verification tool to generate a counter-example for the given assertion that contains the SCV-guided patterns (PI_{hi}).

3.4.3 Noise Power Estimation. For evaluating the SCV metric, we need to estimate the average noise power (P_{noise}). One possible approach is to simulate the crypto module for multiple plaintexts and calculate the average total power. Then P_{signal} can be deducted to retrieve P_{noise}. However, this approach requires us to simulate the crypto module for many plaintexts, which is not feasible from assessment time perspective in pre-silicon stage (as discussed in Section 2).

To overcome this problem, we utilize a vector-less power estimation technique. The vector-less power analysis propagates the signal probability and toggle rates from primary inputs to the outputs of internal nodes and repeats the operation until the primary outputs are reached and all nodes are assigned an activity rate. The derived activity rates are then used to compute power consumption numbers [6]. To run the vector-less power estimation technique, the verification engineer needs to define the signal probability and toggle rates of primary input ports. Signal probability is defined as the percentage of the analysis during which the input is driven at a high logic level and toggle rate is defined as the rate at which a net or logic element switches compared to its input(s) [1].

We utilize the vector-less power analysis to estimate P_{noise}. To achieve good accuracy, we set the signal rate, static signal probability, and toggle rates to resemble the practical conditions. During an actual SCA, the key will remain static. Therefore, we set toggle rate as zero for all key input pins to mimic this condition. For the plaintext input ports, we consider that each port has equal probability of receiving 0 and 1. This is a valid assumption if we consider a large number of random plaintexts, which is common for side-channel leakage assessment. Therefore, we set the input signal probability to be 0.5. We use the toggle rate of 100%, because in an encryption module, we expect the inputs of the synchronous elements to switch on every clock edge due to the confusion and diffusion effects. In this work, we utilize the PrimeTime tool from Synopsys and XPE tool from Xilinx for the vector-less power analysis for ASIC and FPGA design flow, respectively.

The vector-less power analysis returns the total estimated power P_{total} of the design. To get the P_{noise}, P_{signal} (defined in Equation (2) and derived using SPG technique) is deducted from P_{total}. That is, P_{noise} is given by the following equation:

$$P_{noise} = P_{total} - P_{signal}. \tag{3}$$

Note that SCV provides a side-channel vulnerability measure of a hardware design at pre-silicon stage where a higher SCV value represents more vulnerability to side-channel attacks. However, the significance of the SCV may be difficult to be interpreted by a verification engineer. To address this issue, we also derive the success rate (SR) of power SCAs from the SCV in Section 3.4.4.

3.4.4 Derivation of SR from SCV. In this section, we first present general mathematical analysis for the success rate based on the maximum likelihood estimation [21] and then derive the relationship between SCV and SR. Let us assume that observable side-channel leakage at the post-silicon evaluation stage is defined as

$$L = \{l_j | l_j = \epsilon h_{j|k} + N, j = 1, 2, \ldots, n\}, \tag{4}$$

where $h_{j|k} = HD(f(x_j, k))$ or $HW(f(x_j, k))$, and N is additive Gaussian noise with the mean μ_N and the variance σ_N^2. The probability density function of L given k is

$$f_{L|k}(l) = \frac{1}{\sqrt{2\pi}\sigma_N} e^{-\frac{(l-\epsilon h-\mu_N)^2}{2\sigma_N^2}}. \tag{5}$$

The likelihood function is defined as $\mathcal{L}(k; l) = \frac{1}{n}\sum_{i=1}^{n} \ln f_{L|k}(l_i)$. Based on the maximum likelihood estimation, a guess key of an adversary is selected as follows:

$$\hat{k} = \arg\max_{k \in K} \mathcal{L}(k; l) = \arg\max_{k \in K} \frac{1}{n} \sum_{i=1}^{n} \ln f_{L|k}(l_i). \tag{6}$$

If the guess key $k_g = \hat{k}$ is equal to the correct key k^*, then the side-channel attack is successful. The success rate (SR) is defined as follows:

$$SR = \Pr[k_g = k^*] = \Pr[\mathcal{L}(k^*; l) > \mathcal{L}(\langle \bar{k^*} \rangle; l)], \tag{7}$$

where $\langle \bar{k^*} \rangle$ denotes all wrong keys, i.e., $\{k_1, k_2, \ldots, k_{n_k-1}\}$ excluding k^*. Let $\Delta(k^*, k_j) = \mathcal{L}(k^*; l) - \mathcal{L}(k_j; l) = \frac{1}{n}\sum_{i=1}^{n}[\ln f_{L|k^*}(l_i) - \ln f_{L|k_j}(l_i)], \forall k_j \in \langle \bar{k^*} \rangle$. Since $\ln f_{L|k^*}(l_i) - \ln f_{L|k_j}(l_i)$ for $i = 1, 2, \ldots, n$, is independently and identically distributed, $[\Delta(k^*, k_j) - \mu_{\Delta(k^*,k_j)}]/\sigma_{\Delta(k^*,k_j)}$ has the standard normal distribution by the central limit theorem. The probability that the likelihood function of the correct key is larger than that of a wrong key, $\Pr[\Delta(k^*, k_j) > 0]$ is $\Phi(\frac{\mu_{\Delta(k^*,k_j)}}{\sigma_{\Delta(k^*,k_j)}})$, where $\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-x^2/2} dx$. The overall success rate over all wrong keys of Equation (7) is

transformed as follows [21]:

$$SR = \Pr[\Delta(k^*, k_1) > 0, \ldots, \Delta(k^*, k_{n_k-1}) > 0] = \Phi_{n_k-1}\left(\sqrt{n}\Sigma^{-1/2}\vec{\mu}\right), \quad (8)$$

where $\Phi_{n_k-1}(x)$ is the cumulative function of the (n_k-1)-dimensional standard normal distribution, Σ is the $(n_k-1)\times(n_k-1)$ covariance matrix with elements $s_{ij} = n \cdot Cov(\Delta(k^*, k_i), \Delta(k^*, k_j))$, $i, j = 1, 2, \ldots, n_k - 1$, and $\vec{\mu}$ is the mean vector, $\vec{\mu} = [\mu_{\Delta(k^*,k_1)}, \ldots, \mu_{\Delta(k^*,k_{n_k-1})}]^T$.

Theorem 1. *The element of the mean vector $\vec{\mu}$ in Equation (8) is*

$$\mu_{\Delta k*,k_i} = \mathrm{E}[\Delta(k^*, k_i)] = \frac{1}{2}\frac{\epsilon^2}{\sigma_N^2}\mathrm{E}[(h_{|k^*} - h_{|k_i})^2], \quad (9)$$

for $i = 1, \ldots, n_k - 1$. The ij-th element of Σ in Equation (8) is

$$s_{ij} = \frac{\epsilon^2}{\sigma_N^2}\mathrm{E}[(h_{|k^*} - h_{|k_i})(h_{|k^*} - h_{|k_j})] + \frac{1}{4}\left(\frac{\epsilon^2}{\sigma_N^2}\right)^2\Big[\mathrm{E}[(h_{|k^*} - h_{|k_i})^2(h_{|k^*} - h_{|k_j})^2] - \mathrm{E}[(h_{|k^*} - h_{|k_i})^2]\mathrm{E}[(h_{|k^*} - h_{|k_j})^2]\Big]. \quad (10)$$

The proof of Theorem 1 is given in Section A.1.

In Equations (9) and (10), $\frac{\epsilon^2}{\sigma_N^2}$ is defined as signal-to-noise ratio (SNR) of the side-channel leakage and E[·] terms are referred to as algorithmic confusion coefficients defined in Reference [21]. The confusion coefficient depends on the target function $T = f(x, k)$ affected by an attack model, e.g., the output of the target function is 0 or 1 in a DPA model.

Theorem 2. *Let $L^{(1)}$ denote side-channel leakage at the pre-silicon stage as follows:*

$$L^{(1)} = \left\{l_j^{(1)} | l_j^{(1)} = \epsilon_1 h_{j|k} + N^{(1)} \ \text{ for } \ j = 1, 2, \ldots, n\right\}, \quad (12)$$

where $h_{j|k} = HD(f(x, k))$ or $HW(f(x, k))$, and $N^{(1)}$ is additive Gaussian noise, denoted by $\mathcal{N}(\mu_{N^{(1)}}, \sigma^2_{N^{(1)}})$ excluding external (or measurement) noise that is observable at the post-silicon evaluation stage, denoted by $N^{(2)} \sim \mathcal{N}(\mu_{N^{(2)}}, \sigma^2_{N^{(2)}})$. SCV_n using the n bit-width target function is related to SNR in Equations (9) and (10) as follows:

$$SCV_n \simeq \frac{n}{\left(1 - \frac{\sigma^2_{N^{(2)}}}{\sigma^2_N}\right)}\alpha^2 SNR, \quad (13)$$

where α^2 is the scaling factor that is equal to $\frac{\epsilon_1}{\epsilon}$ and SNR is $\frac{\epsilon^2}{\sigma_N^2}$. The proof of Theorem 2 is given in Section A.2.

Equation (12) establishes the mathematical relationship between the SNR and SCV. This equation also provides the theoretical explanation of the experimental observation that SCV (estimated at pre-silicon by SCRIPT) and SNR (experimentally evaluated at post-silicon) are highly correlated and only differ by a scaling factor (see Section 4.4). In addition, Equation (12) allows us to estimate $\frac{\epsilon^2}{\sigma_N^2}$ of Equation (10) from SCV and derive the SR at pre-silicon. SR represents the number of plaintexts required to extract a certain percentage of the correct key and, therefore, provides a quantitative measure of SCA vulnerability.

Table 2. Metric Notations and Platforms to Which They Are Applied in Our Experimental Results

Level	Platform	PD	KL	SCV	SNR	SR
Pre-silicon	ASIC-SIM	✓	×	✓	×	✓
	FPGA-SIM	✓	×	✓	×	✓
Post-silicon	FPGA-EXP	×	✓	×	✓	✓

PD, KL, SNR, and SR refer to power difference, KL-divergence, signal-to-noise ratio, and success rate, respectively. The tick marks represent the platforms to which these metrics are applied.

4 EXPERIMENTAL RESULTS AND DISCUSSION

In this section, we perform PSCL assessment of two different implementations of AES algorithm using SCRIPT framework. First, we provide a brief description of the two AES designs: AES Galois Field (AES-GF) and AES lookup table (AES-LUT). Then, we present the results generated by each module in the SCRIPT framework and validate the collected results. We validate the effectiveness of an SPG engine using experimental results obtained from FPGAs. Finally, we compare SCV (estimated at pre-silicon by SCRIPT) with SNR (evaluated at post-silicon) in terms of correlation coefficient. We also validate that SCRIPT framework can effectively estimate SCA success rate (SR) using post-silicon results.

Table 2 presents the notations of the metrics used in the following analyses as well as the platforms to which these metrics are applied. Here, ASIC-SIM and FPGA-SIM refer to SCRIPT simulation results using ASIC and FPGA flows in Figure 4, while FPGA-EXP refers to the actual silicon results from experimental analysis on Xilinx FPGAs.

All our pre-silicon experiments were performed on a 64-bit Linux system running on Intel Core i5 at 1.800 GHz with 8 GB of memory. For post-silicon (FPGA) validation (expressed as FPGA-EXP), we use the SAKURA-G board, which contains two Xilinx SPARTAN-6 FPGAs to implement the hardware designs. Tektronix MDO3102 oscilloscope is used to measure the voltage drop between a shunt resistor connected to the Vdd pin. The clock frequency of the AES implementation is 24 MHz, and the sampling rate and bandwidth of the oscilloscope are 500 MS/s and 250 MHz, respectively.

4.1 AES Benchmarks

The SCRIPT framework for PSCL assessment is applied to AES-GF [3] and AES-LUT [4] implementations. Both are open-source designs. Figure 3 shows the hardware architecture of the AES-GF encryption module. In this architecture, the AES round operations and the AES key expansion operation occurs in parallel. AES-GF architecture takes 10 clock cycles to encrypt each data block. The main characteristic of this design is that it implements the AES "SubByte" operation using Galois field arithmetic.

AES-LUT design, however, first performs the key expansion and stores the expanded keys in the key registers. After the key expansion, the round operation starts and takes 10 cycles to perform each encryption. AES-LUT design implements AES "SubByte" operation using a look-up table.

4.2 Results: Target Registers Identification

The IFT engine in SCRIPT works at the gate-level netlist. Both AES-GF and AES-LUT designs are provided in Verilog RTL format. We first synthesize the RTL designs into a gate-level netlist using Synopsys Design Compiler [5] with Synopsys standard cell library. The Target register identification for AES-GF design is discussed in detail in Section 3.3.3. We then perform the same IFT

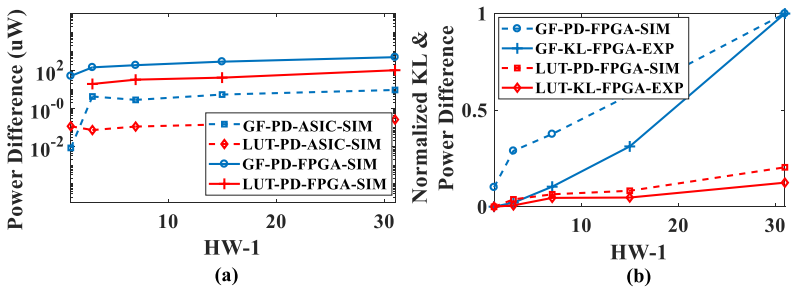

Fig. 5. (a) Power difference vs. difference in HW between ASIC-SIM and FPGA-SIM using a base 10 logarithmic scale for the y-axis. (b) Normalized value of power difference in FPGA-SIM and the normalized value of KL-divergence in FPGA-EXP vs. HW difference.

analysis for AES-LUT and identify the target registers. We also find *Data_Reg*[0 *to* 31] as target registers for AES-LUT. The reason is that both designs have similar architecture for round operations.

4.3 Results: SCV-guided Pattern Generation

SCRIPT utilizes the SPG engine to produce patterns/plaintexts with the desired Hamming weight (HW) of the target registers and calculate the SCV metric. The power difference (PD) generated by different HWs of the target registers is exploited for the DPA and CPA attacks. In this section, we utilize SCRIPT framework to generate patterns for six different HWs and present the power difference for the corresponding HWs for both ASIC-SIM and FPGA-SIM flow (as shown in Figure 4). We validate that the "simulated" power difference derived by SCV-guided patterns at the gate level has a high correlation to the power difference "experimentally" measured on FPGAs. We will also present silicon results demonstrating that the SPG pattern can generate much greater power difference than those generated by random patterns/plaintexts.

We first utilize SCRIPT framework to generate patterns for six different $HW = 1, 2, 4, 8, 16, 32$. Figure 5(a) plots $HW - 1$ vs. power difference using a base 10 logarithmic scale for the y-axis and a linear scale for the x-axis. For both AES-GF and AES-LUT implementations in both ASIC-SIM and FPGA-SIM, power difference increases almost linearly. This observation conforms with the power hypothesis model, which states that power consumption is proportional to HW [33]. Figure 5(a) also illustrates that the power difference in AES-GF is much higher when compared to AES-LUT, which means that AES-GF leaks more side-channel information than AES-LUT. This observation is consistent in both ASIC-SIM and FPGA-SIM flows. Also, the power consumption of ASIC-SIM is much lower as compared to FPGA-SIM, which is expected, since FPGAs in general have larger power requirements.

One logical question that may arise is, why do we evaluate up to 32-bit HW when an attacker in general attacks at most 8-bit of the target function? While this is true from an attack perspective, from PSCL point of view, we want to measure the worst-case leakage of the design. Therefore, we measure up to 32-bit HW, which represents the worst-case leakage.

4.3.1 FPGA Validation: SCV-guided Pattern Generation. Figure 6 shows the experimental setup for FPGA validation. We use the same setup for all our FPGA results presented in this article.

In addition, we apply our SCV-guided patterns/plaintexts and collect their power signals corresponding to the HWs and then calculate the KL-divergence between the resulting power signal distributions. Figure 7 shows the KL-divergence between $HW = 1$ and $HW = i$ (denoted by $KL(HW = 1|HW = i)$, where $i = 2, 4, 8, 16, 32$ for AES-GF and AES-LUT implementations in

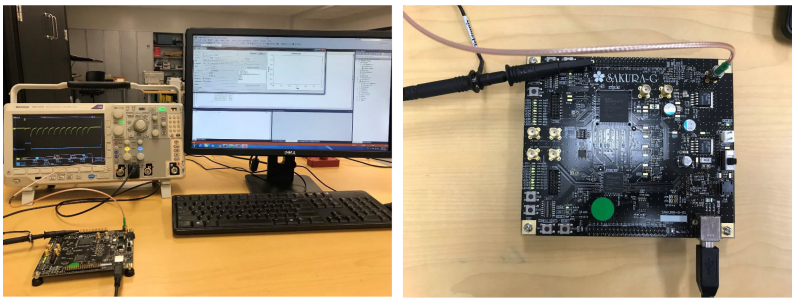

Fig. 6. Experimental setup for FPGA validation.

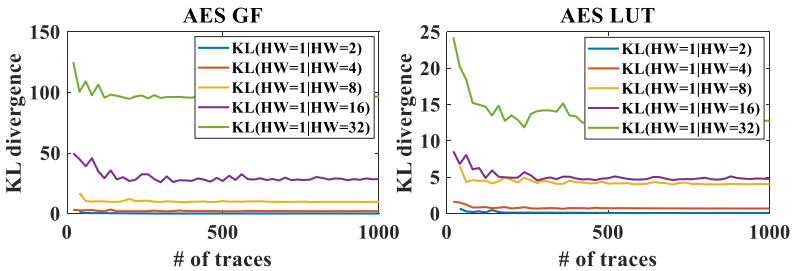

Fig. 7. KL-divergence of $HW = 1$ and $HW = i$ where $i = 2, 4, 8, 16, 32$ for AES-GF and AES-LUT implementations in FPGA-EXP.

FPGA-EXP. It shows that the KL-divergence increases linearly with i. Note that KL-divergence is traditionally used for power side-channel leakage comparison where a higher KL-divergence value corresponds to higher leakage.

Figure 5(b) shows the normalized value of FPGA power differences and the normalized value of $KL(HW = 1|HW = i)$. The figure shows a good correlation between our simulated (SIM) power difference and experimentally measured (EXP) KL-divergence. The Pearson correlation coefficient for AES-GF is 0.96, whereas for AES-LUT it is 0.98. This result validates that the power difference in FPGA-SIM evaluated by the SPG engine at gate level is highly correlated with the KL divergence in FPGA-EXP.

We also use random patterns to estimate the KL-divergence using the procedure described in Reference [38]. The KL-divergence using random patterns is 3.3 and 2.5 for AES-GF and AES-LUT implementation in FPGA-EXP, respectively. This is 30 times and 7 times less compared to SCV-guided patterns $HW = 32$, respectively. KL-divergence can be used to distinguish vulnerable implementations [38]. From this perspective, one can see that our SCV-guided patterns produce much better results as compared to the random patterns.

4.4 Results: SCV Estimation and Validation

In this section, we present our SCV metric calculated by SCRIPT. Once we have calculated the power difference for our desired HW, we measure the P_{noise} using the vector-less power estimation technique (discussed in Section 3.4.3). We then calculate SCV values in ASIC-SIM and FPGA-SIM flows (shown in Figure 4). Figure 8(a) shows $HW - 1$ vs. SCV values in ASIC-SIM and FPGA-SIM platforms using a base 10 logarithmic scale for the y-axis and a linear scale for the x-axis. According to the SCV metric, the AES-GF implementation is more vulnerable to side-channel leakage as compared to the AES-LUT. Also, note that the SCV metric is greater in ASIC-SIM w.r.t. FPGA-SIM.

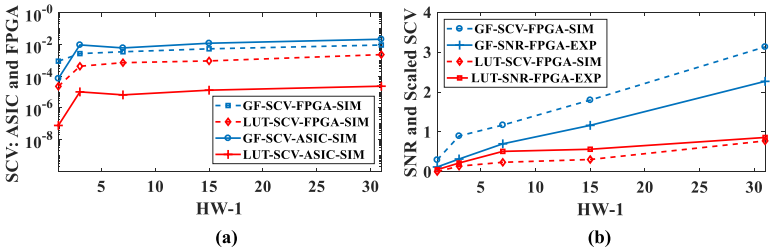

Fig. 8. (a) *SCV* metric vs. difference in HW between ASIC-SIM and FPGA-SIM using a base 10 logarithmic scale for the y-axis. (b) Scaled *SCV* value vs. difference in HW in FPGA-SIM and *SNR* metric in FPGA-EXP.

Table 3. Comparison: Area (in Terms of Number of Gates (**G**), LUTs (**L**), and Registers (**R**)), Performance (in Terms of Maximum Delay of the Design) vs. Security (in Terms of Maximum SCV)

Platform	Design	Area	Timing	*SCV*
ASIC-SIM	AES-GF	11,407 **G**; 796 **R**	89.5 ns	$2.1 * 10^{-2}$
	AES-LUT	2,695 **G**; 650 **R**	94.3 ns	$2.3 * 10^{-5}$
FPGA-SIM	AES-GF	3,931 **L**; 796 **R**	5.9 ns	$9.4 * 10^{-3}$
	AES-LUT	2,480 **L**; 650 **R**	5.0 ns	$2.3 * 10^{-3}$

The reason is that P_{noise} in ASIC-SIM is lower as compared to FPGA-SIM (FPGA has higher power consumption).

Next, we validate SCV metric estimated at pre-silicon by calculating the SNR metric experimentally from FPGA. We evaluate the SNR metric in FPGA-EXP for different HWs using the same measurement technique shown in References [21, 33]. As discussed in Section 3.4, SCV is conceptually similar to SNR; however, due to the different calculation strategies, these two values are not numerically equal. Figure 8(b) shows the scaled values of SCV metric in FPGA-SIM and SNR metric measured in FPGA-EXP. It shows there exists good correlation between these two metrics. The Pearson correlation coefficient for AES-GF is 0.99, whereas for AES-LUT it is 0.94. This validates that our SCV metric can be used to obtain a good estimate of the SNR. We empirically derive the scaling factor to be $(0.003)^{-1}$. Note that the scaling factor is common for all designs, e.g., the SCV values for both AES-GF and AES-LUT are scaled with the same factor (shown in Figure 8(b)). In Appendix B, we mathematically derive the expression for scaling factor.

Evaluation Time: The SCRIPT framework applied to the gate-level AES designs requires on average 3 minutes for the IFT analysis and less than 1 minute for the SPG. This step is common for both ASIC-SIM and FPGA-SIM flow. The SCV-guided pattern simulation and power estimation on average require less than 1 minute for ASIC-SIM and 5 minutes for FPGA-SIM (per pattern). Therefore, the overall runtime (on average) for SCV estimation requires 6 minutes for ASIC-SIM and 14 minutes for FPGA-SIM. The required evaluation time is much smaller than previously proposed pre-silicon-based PSCL assessment [19, 41]. For example, if we want to evaluate SNR metric using 10K plaintext inputs in the gate-level simulation, it would take around 31 days to measure it (it takes on average 4.5 minutes to estimate power for each pattern in our simulation using similar workstation).

Area, Performance vs. Security: Table 3 shows the comparison between area, performance, and security for AES-GF and AES-LUT designs for both ASIC-SIM and FPGA-SIM flows. Area is represented in terms of the number of gates/LUTs and registers, whereas, performance is represented in terms of maximum delay. Security is represented in terms of maximum SCV value.

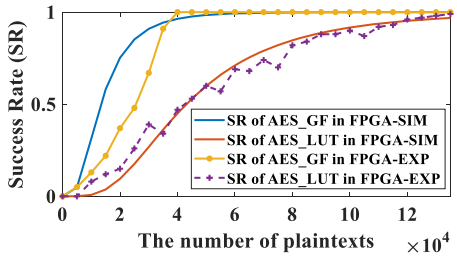

Fig. 9. SR estimated in FPGA-SIM and SR calculated in FPGA-EXP of AES-GF and AES-LUT implementations w.r.t. number of plaintexts.

Table 3 states that this particular AES-GF implementation requires relatively higher area as compared to the AES-LUT implementation. Also, AES-GF implementation is more vulnerable to power SCA compared to AES-LUT implementation, as indicated by the SCV metric.

SCV vs. SNR: SCRIPT utilizes formal verification to generate guided patterns. These patterns only introduce switching in the logic related to the target function while muting the switching of the rest of the design and, thereby, derive SCV using as few as two patterns. These two patterns produce maximum power difference in the Target function. In contrast, SNR is calculated by performing DPA, which requires thousands of plaintexts to make the difference of mean (DOM), i.e., the numerator of the SNR equation measurable [35]. Therefore, evaluating SNR metric at pre-silicon (using 10K plaintext) would require 31 days, whereas SCV metric requires 14 minutes to evaluate using same simulation setup.

4.5 Success Rate: SCV and SNR

As mentioned in Section 3.4.4, SR can be estimated using SNR and the algorithmic confusion coefficient. Since the algorithmic confusion coefficient is determined by the target function and the SNR is estimated by the SCV (SCV is related to SNR by a scaling factor), the SCRIPT framework allows us to estimate SR w.r.t. the number of plaintexts (or leakage power traces) at pre-silicon design stage. We estimate SNRs of 0.016 and 0.0075 for AES-GF and AES-LUT implementations, respectively, using SCV and the aforementioned empirically derived scaling factor. By Equation (8), SR is estimated in FPGA-SIM for AES-GF and AES-LUT and is illustrated in Figure 9. To validate the accuracy of the estimated SR in FPGA-SIM, CPA attacks are experimentally performed on FPGA-EXP to reveal an 8-bit sub-key. In our experiments, 100 CPA attacks with randomly generated n plaintexts are performed repeatedly in FPGA-EXP and the number of successful attacks are counted to measure the SR w.r.t. n plaintexts in FPGA-EXP. At least 40K and 130K plaintexts are required to satisfy 100% SR in AES-GF and AES-LUT implementations, respectively, in FPGA-EXP. Figure 9 shows the plots of SR estimated at FPGA-SIM and SR calculated in FPGA-EXP of AES-GF and AES-LUT implementations w.r.t. the number of plaintexts. The Pearson correlation coefficients between the SR estimated at FPGA-SIM and the SR calculated in FPGA-EXP of AES-GF and AES-LUT are 0.93 and 0.99, respectively. Therefore, this demonstrates that SCRIPT accurately predicts the SR (or equivalently the number of plaintexts required for a successful SCA) in pre-silicon stage.

4.6 PSCL Assessment of Non-crypto Design

Non-crypto hardware designs can possess the properties of the target function and, therefore, can also be vulnerable to SCAs. Established strategies for PSCL assessment of crypto designs may not be applicable to non-crypto designs. SCRIPT framework can be a viable solution to identify

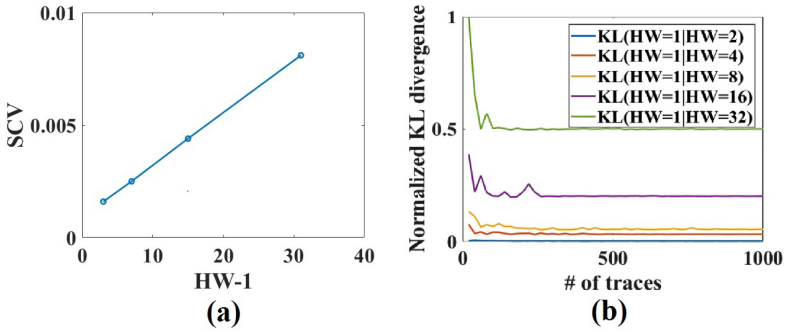

Fig. 10. (a) SCV of FPGA-SIM w.r.t. difference in HW for RISC-MULT, (b) KL-divergence of FPGA-EXP for $HW = 1$ and $HW = i$ where $i = 2, 4, 8, 16, 32$ for RISC-MULT in FPGA-EXP.

potential PSCL vulnerabilities in such designs. This subsection presents PSCL assessment of a 32x32 multiplication unit of a RISC processor (represented as RISC-MULT) using SCRIPT. Note that multiplication operations have been exploited by SCAs. Aysu et al. [9] presented a side-channel attack on a post-quantum key exchange protocol by exploiting the multiplication operation that depends on the subkey. However, the PSCL assessment methodology for non-crypto designs like multiplication units has not been investigated.

We first mark the multiplicand port as a key and the multiplier port as an input that an attacker can control. We then apply SCRIPT's IFT analysis on RISC-MULT, which returns that the registers storing the intermediate multiplication values exhibit the P1-P4 properties of the target function, and, therefore, RISC-MULT is vulnerable to SCAs. We then utilize SCRIPT's SPG engine to generate patterns for six different $HW = 1, 2, 4, 8, 16, 32$ and calculate a SCV metric for each HW. Figure 10(a) shows $HW - 1$ vs. SCV values for RISC-MULT-FPGA-SIM. For FPGA validation, we apply the same patterns in FPGA implementation of RISC-MULT and collect their power signals corresponding to the HWs and then calculate the KL-divergence between the resulting power signal distributions. Figure 10(b) shows the normalized KL-divergence between $HW = 1$ and $HW = i$ (denoted by $KL(HW = 1|HW = i)$, where $i = 2, 4, 8, 16, 32$ for RISC-MULT-FPGA-EXP, which shows a good correlation between SCV and experimentally measured KL-divergence. The Pearson correlation coefficient is 0.995. It validates that the SCV metric evaluated by SCRIPT at FPGA-SIM is highly correlated with the KL divergence in FPGA-EXP. Therefore, SCRIPT can be an effective framework for analyzing PSCL of non-crypto hardware designs.

5 FUTURE WORK AND CONCLUSION

While the SCRIPT framework paves the way for automated PSCL assessment at pre-silicon stage, there are some limitations in the framework and room for future improvements. The SCRIPT framework in its current version can only track target registers. However, we plan to extend our IFT technique to detect nets as well, to increase the granularity of our IFT analysis and identify intermediate target operations.

Another limitation of the SCRIPT framework is that it may not be used to assess SCA resiliency for certain hardware designs incorporating masking countermeasures. The reason is that masking utilizes randomization to counteract SCAs, which is not compatible with SCRIPT. We plan to address this limitation by extending SCRIPT's compatibility to work with random number generators. Also, in this article, we focus on the first-order side-channel attacks. In our future work, we plan to extend SCRIPT's scope to include the higher-order SCAs as well.

SCRIPT framework can identify which specific operation and at which clock instance leaks side-channel information. This knowledge can be utilized to apply localized countermeasures. That is, instead of blindly applying a countermeasure to the whole design, a countermeasure for only operations that are responsible for PSCL can be developed. Such localized countermeasure techniques would require lower cost and performance overhead. Our future plan includes utilizing SCRIPT framework to develop countermeasures that only focus on target functions.

SCRIPT framework can be an important component of the recent academic and industrial initiatives for developing CAD frameworks that aim at automating the security vulnerability assessment of hardware designs at design stages [37, 44]. This framework would allow the semiconductor industry to systematically and efficiently identify side-channel vulnerabilities before tape-out to include proper countermeasures or refine the design to address them.

A APPENDIX

A.1 Proof of Theorem1: Success Rate

$$\begin{aligned} SR &= \Pr[\Delta(k^*, k_1) > 0, \ldots, \Delta(k^*, k_{n_k-1}) > 0] \\ &= \Phi_{n_k-1}\left(\sqrt{n}\Sigma^{-1/2}\vec{\mu}\right), \end{aligned} \tag{14}$$

where $\Phi_{n_k-1}(x)$ is the cumulative function of the $(n_k - 1)$-dimensional standard normal distribution, Σ is the $(n_k - 1) \times (n_k - 1)$ covariance matrix with elements $s_{ij} = n \cdot Cov(\Delta(k^*, k_i), \Delta(k^*, k_j))$, $i, j = 1, 2, \ldots, n_k - 1$, and $\vec{\mu}$ is the mean vector, $\vec{\mu} = [\mu_{\Delta(k^*,k_1)}, \ldots, \mu_{\Delta(k^*,k_{n_k-1})}]^T$. In Equation (13),

$$\begin{aligned} \Delta(k^*, k_i) &= \frac{1}{n}\sum_{j=1}^{n}\left[\ln f_{L|k^*}(l_j) - \ln f_{L|k_i}(l_j)\right] \\ &= \frac{1}{2\sigma_N^2 n}\sum_{j=1}^{n}\left[(l_j - \epsilon h_j^i - \mu_N)^2 - (l_j - \epsilon h_j^* - \mu_N)^2\right]. \end{aligned}$$

Let $r_j = l_j - \mu_N - \epsilon h_j^*$. r_j is a Gaussian random variable, i.e., $r_j \sim \mathcal{N}(0, \sigma_N^2)$. Then,

$$\begin{aligned} \Delta(k^*, k_j) &= \frac{1}{2\sigma_N^2 n}\sum_{j=1}^{n}\left[(r_j + \epsilon(h_j^* - h_j^i))^2 - r_j^2\right] \\ &= \frac{\epsilon^2}{2n\sigma_n^2}\sum_{j=1}^{n}\left[(h_j^* - h_j^i)^2 + \frac{2}{\epsilon}r_j(h_j^* - h_j^i)\right]. \end{aligned}$$

Since $\mathrm{E}[r_j] = 0$, and r_j and $(h_j^* - h_j^i)$ are independent, $\mathrm{E}[r_j(h_j^* - h_j^i)] = 0$. Thus,

$$\mathrm{E}[\Delta(k^*, k_j)] = \frac{1}{2n}\left(\frac{\epsilon^2}{\sigma_N^2}\right)n\mathrm{E}[(h_j^* - h_j^i)^2] = \frac{1}{2}\left(\frac{\epsilon^2}{\sigma_N^2}\right)\mathrm{E}[(h_{|k^*} - h_{|k_j})^2].$$

The ij-th element of Σ in Equation (13) is $s_{ij} = n \cdot Cov(\Delta(k^*, k_i), \Delta(k^*, k_j))$.

$$
\begin{aligned}
Cov(\Delta(k^*,k_i),\Delta(k^*,k_j)) &= \mathrm{E}[\Delta(k^*,k_i)\Delta(k^*,k_j)] \\
&\quad - \mathrm{E}[\Delta(k^*,k_i)\mathrm{E}[\Delta(k^*,k_j)] \\
&= \left(\frac{\epsilon^2}{2n\sigma_N^2}\right)^2 \sum_{m=1}^{n}\sum_{l=1}^{n} \mathrm{E}\left\{[(h_l^* - h_l^i)^2 + \frac{2}{\epsilon}(h_l^* - h_l^i)r_l] \right. \\
&\quad \left. \cdot[(h_m^* - h_m^j)^2 + \frac{2}{\epsilon}(h_m^* - h_m^j)r_m]\right\} - \mathrm{E}[\Delta(k^*,k_i)]\mathrm{E}[\Delta(k^*,k_j)].
\end{aligned}
$$

Since $\mathrm{E}[r_l] = \mathrm{E}[r_m] = 0$, and $\mathrm{E}[r_l r_m] = 0$ for $l \neq m$,

$$
\begin{aligned}
Cov(\Delta(k^*,k_i),\Delta(k^*,k_j)) &= \left(\frac{\epsilon^2}{2n\sigma_N^2}\right)^2 \sum_{m=1}^{n}\sum_{l=1}^{n} \mathrm{E}[(h_l^* - h_l^i)^2(h_m^* - h_m^j)^2] \\
&\quad + \sum_{l=1}^{n}\left(\frac{2}{\epsilon}\right)^2 \sigma_N^2 \mathrm{E}[(h_l^* - h_l^i)][(h_l^* - h_l^j)] - \mathrm{E}[\Delta(k^*,k_i)]\mathrm{E}[\Delta(k^*,k_j)] \\
&= \frac{1}{n}\left\{\left(\frac{\epsilon}{\sigma_N}\right)^2 \mathrm{E}[(h_{|k^*} - h_{|k_i})(h_{|k^*} - h_{|k_j})] \right. \\
&\quad + \frac{1}{4}\left(\frac{\epsilon^2}{\sigma_N^2}\right)^2 \left[\mathrm{E}[(h_{|k^*} - h_{|k_i})^2(h_{|k^*} - h_{|k_j})^2] \right. \\
&\quad \left. \left. - \mathrm{E}[(h_{|k^*} - h_{|k_i})^2]\mathrm{E}[(h_{|k^*} - h_{|k_j})^2]\right]\right\}.
\end{aligned}
$$

A.2 Proof of Theorem2: SCV Metric

Let $L^{(1)}$ be the leakage at the pre-silicon stage as follows: $L^{(1)} = \{l_j | l_j^{(1)} = \epsilon_1 h_j + N^{(1)}\}$. SCV is defined as follows:

$$
SCV = \frac{P_{signal}}{P_{noise}} = \frac{P_{T.hi} - P_{T.hj}}{P_{noise}}. \tag{15}
$$

Let P_{signal} in Equation (14) be the power consumption of the target function with n Hamming distance as follows:

$$
P_{signal} = n\epsilon_1^2. \tag{16}
$$

Let P_{noise} in Equation (14) be the power consumption caused by the switching activity of other functions as follows:

$$
P_{noise} = \mathrm{E}[\{N^{(1)}\}^2] = \mathrm{Var}[N^{(1)}] + \{\mathrm{E}[N^{(1)}]\}^2. \tag{17}
$$

Since $\mathrm{E}[N^{(1)}]$ is equal to $\frac{w_h}{2}\epsilon_1$, where w_h is the bit-width of other functions (i.e., the output transition of other functions is equally distributed), P_{noise} is replaced as follows:

$$
P_{noise} = \sigma_{N^{(1)}}^2 + \left(\frac{w_h}{2}\epsilon_1\right)^2 = \sigma_N^2 - \sigma_{N^{(2)}}^2 + \left(\frac{w_h}{2}\epsilon_1\right)^2, \tag{18}
$$

where N is the total noise including external (or measurement) noise, $N^{(2)}$ in a measured leakage such that $N = N^{(1)} + N^{(2)}$ (since $N^{(1)}$ and $N^{(2)}$ are independent, $\sigma_N^2 = \sigma_{N^{(1)}}^2 + \sigma_{N^{(2)}}^2$). By Equations (15) and (17), SCV_n using the n-bit target function is related to SNR as follows:

$$
\begin{aligned}
SCV_n = \frac{P_{signal}}{P_{noise}} &= \frac{n\epsilon_1^2}{\sigma_N^2 - \sigma_{N^{(2)}}^2 + \left(\frac{w_h}{2}\epsilon_1\right)^2} \\
&= \frac{n\frac{\epsilon_1^2}{\epsilon_1^2}}{\frac{1}{\epsilon_1^2}(\sigma_N^2 - \sigma_{N^{(2)}}^2) + \left(\frac{w_h}{2}\frac{\epsilon_1}{\epsilon_1}\right)^2} \\
&= \frac{n}{\left(\frac{w_h^2}{4}\right) + \frac{\sigma_N^2}{\epsilon_1^2}\left(1 - \frac{\sigma_{N^{(2)}}^2}{\sigma_N^2}\right)} \\
&= \frac{n}{\left(\frac{w_h^2}{4}\right) + \frac{\sigma_N^2}{\epsilon^2}\frac{\epsilon^2}{\epsilon_1^2}\left(1 - \frac{\sigma_{N^{(2)}}^2}{\sigma_N^2}\right)} \\
&= \frac{n}{\left(\frac{w_h^2}{4}\right) + \frac{1}{\frac{\epsilon^2}{\sigma_N^2}\frac{\epsilon_1^2}{\epsilon^2}}\left(1 - \frac{\sigma_{N^{(2)}}^2}{\sigma_N^2}\right)} \\
&= \frac{n}{\left(\frac{w_h^2}{4}\right) + \frac{1}{\alpha^2 SNR}\left(1 - \frac{\sigma_{N^{(2)}}^2}{\sigma_N^2}\right)} \\
&\simeq \frac{n}{\left(1 - \frac{\sigma_{N^{(2)}}^2}{\sigma_N^2}\right)}\alpha^2 SNR,
\end{aligned}
$$

where α^2 is the scaling factor, which is equal to $\frac{\epsilon_1}{\epsilon}$ and SNR is $\frac{\epsilon^2}{\sigma_N^2}$. And, generally, $\frac{w_h^2}{4} << \frac{1}{\alpha^2 SNR}(1 - \frac{\sigma_{N^{(2)}}^2}{\sigma_N^2})$

A.3 List of Abbreviations in This Article

SCRIPT - Side-Channel vulneRability using Information flow tracking and PaTtern gereration.
SCA - Side-Channel Attacks.
CAD - Computer-Aided Design.
PSCL - Power Side-Channel Leakage.
SCV - Side-Channel Vulnerability.
SNR - Signal-to-Noise Ratio.
IoT - Internet of Things.
DPA - Differential Power Analysis.
CPA - Correlation Power Analysis.
MIA - Mutual Information Attack.
PPA - Partitioning Power Analysis.
EDA - Electronic Design Automation.
ASIC - Application-Specific Integrated Circuit.
FPGA - Field Programmable Gate Array.
IFT - Information Flow Tracking.
SPG - SCV-guided Pattern Generation.
SR - Success Rate.
GF - Galois Field.
LUT - Look-Up Table.

HW - Hamming Weight.
HD - Hamming Distance.
DOM - Difference-Of-Means.
ATPG - Automatic Test Pattern Generation.
FF - Flip-Flop.
SAIF - Switching Activity Interchange Format.
ASIC-SIM - ASIC SIMulation.
FPGA-SIM - FPGA SIMulation.
FPGA-EXP - FPGA EXPeriment.
PD - Power Difference.
KL - Kullback Leibler.

REFERENCES

[1] Xilinx, Inc. 2019. Power Analysis and Optimization. https://www.xilinx.com/.
[2] Cadence Design Systems, Inc. 2019. Cadence. https://www.cadence.com/.
[3] Tohoku University. 2019. Galois field based AES verilog design. http://www.aoki.ecei.tohoku.ac.jp/.
[4] Satoh Laboratory. 2019. Lookup table based AES verilog design. *Satoh Laboratory UEC*. http://satoh.cs.uec.ac.jp/en/.
[5] Synopsys. 2019. Synopsys. http://www.synopsys.com/.
[6] Xilinx, Inc. 2019. Vectorless Estimation. https://www.xilinx.com/support/documentation/.
[7] Xilinx, Inc. 2019. Xilinx. https://www.xilinx.com.
[8] Martin Aigner, Stefan Mangard, Francesco Menichelli, Renato Menicocci, Mauro Olivieri, Thomas Popp, Giuseppe Scotti, and Alessandro Trifiletti. 2006. Side channel analysis resistant design flow. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'06)*.
[9] Aydin Aysu, Youssef Tobah, Mohit Tiwari, Andreas Gerstlauer, and Michael Orshansky. 2018. Horizontal side-channel vulnerabilities of post-quantum key exchange protocols. In *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST'18)*. IEEE, 81–88.
[10] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. 2015. Verified proofs of higher-order masking. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 457–485.
[11] Ali Galip Bayrak, Francesco Regazzoni, Philip Brisk, François-Xavier Standaert, and Paolo Ienne. 2011. A first step towards automatic application of power analysis countermeasures. In *Proceedings of the 48th Design Automation Conference*. ACM, 230–235.
[12] Ali Galip Bayrak, Francesco Regazzoni, David Novo, Philip Brisk, François-Xavier Standaert, and Paolo Ienne. 2013. Automatic application of power analysis countermeasures. *IEEE Trans. Comput.* 64, 2 (2013), 329–341.
[13] Ali Galip Bayrak, Francesco Regazzoni, David Novo, and Paolo Ienne. 2013. Sleuth: Automated verification of software power analysis countermeasures. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 293–310.
[14] Roderick Bloem, Hannes Gross, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter. 2018. Formal verification of masked hardware implementations in the presence of glitches. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 321–353.
[15] Eric Brier, Christophe Clavier, and Francis Olivier. 2004. Correlation power analysis with a leakage model. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 16–29.
[16] Michael Bushnell and Vishwani Agrawal. 2004. *Essentials of Electronic Testing for Digital, Memory, and Mixed-signal VLSI Circuits*. Vol. 17. Springer Science & Business Media.
[17] Gustavo K. Contreras, Adib Nahiyan, Swarup Bhunia, Domenic Forte, and Mark Tehranipoor. 2017. Security vulnerability analysis of design-for-test exploits for asset protection in SoCs. In *Proceedings of the 22nd Asia and South Pacific Design Automation Conference (ASP-DAC'17)*. IEEE, 617–622.
[18] Jerry den Hartog, Jan Verschuren, E. de Vink, Jaap de Vos, and W. Wiersma. 2003. PINPAS: A tool for power analysis of smartcards. In *Proceedings of the IFIP International Information Security Conference*. Springer, 453–457.
[19] Yaseer Arafat Durrani and Teresa Riesgo. 2014. Power estimation for intellectual property-based digital systems at the architectural level. *J. King Saud Univ.—Comput. Inf. Sci.* 26, 3 (2014), 287–295. DOI: https://doi.org/10.1016/j.jksuci.2014.03.005
[20] François Durvaux and François-Xavier Standaert. 2016. From improved leakage detection to the detection of points of interests in leakage traces. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 240–262.

[21] Yunsi Fei, A. Adam Ding, Jian Lao, and Liwei Zhang. 2014. A statistics-based fundamental model for side-channel attack analysis. *IACR Cryptology ePrint Archive* (2014), 152.
[22] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. 2008. Mutual information analysis. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 426–442.
[23] Benedikt Gierlichs, Kerstin Lemke-Rust, and Christof Paar. 2006. Templates vs. Stochastic Methods. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 15–29.
[24] Benjamin Jun Gilbert Goodwill, Josh Jaffe, Pankaj Rohatgi, et al. 2011. A testing methodology for side-channel resistance validation. In *Proceedings of the NIST Non-invasive Attack Testing Workshop*.
[25] Z. Hanna. 2013. Verifying security aspects of SoC designs with Jasper app. *(white paper), Jasper Design Automation (Cadence)* (2013). https://www.cadence.com/en_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-gold-verification-platform/security-path-verification-app.html.
[26] Wei Hu, Dejun Mu, Jason Oberg, Baolei Mao, Mohit Tiwari, Timothy Sherwood, and Ryan Kastner. 2014. Gate-level information flow tracking for security lattices. *ACM Trans. Des. Autom. Electron. Syst.* 20, 1 (2014), 1–25.
[27] Sorin A. Huss and Oliver Stein. 2017. A novel design flow for a security-driven synthesis of side-channel hardened cryptographic modules. *J. Low Power Electron. Applic.* 7, 1 (2017), 4.
[28] Sorin A. Huss, Marc Stöttinger, and Michael Zohner. 2013. AMASIVE: An adaptable and modular autonomous side-channel vulnerability evaluation framework. In *Number Theory and Cryptography*. Springer, 151–165.
[29] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential power analysis. In *Proceedings of the International Cryptology Conference*. Springer, 388–397.
[30] Thanh-Ha Le, Jessy Clédière, Cécile Canovas, Bruno Robisson, Christine Servière, and Jean-Louis Lacoume. 2006. A proposition for correlation power analysis enhancement. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 174–186.
[31] Qiasi Luo and Yunsi Fei. 2011. Algorithmic collision analysis for evaluating cryptographic systems and side-channel attacks. In *Proceedings of the IEEE International Symposium on Hardware-oriented Security and Trust (HOST'11)*. IEEE, 75–80.
[32] Stefan Mangard. 2004. Hardware countermeasures against DPA—A statistical analysis of their effectiveness. In *Cryptographers' Track at the RSA Conference*. Springer, 222–235.
[33] Thomas S. Messerges, Ezzat A. Dabbish, and Robert H. Sloan. 2002. Examining smart-card security under the threat of power analysis attacks. *IEEE Trans. Comput.* 51, 5 (2002), 541–552.
[34] Amir Moradi. 2006. Masking as a side-channel countermeasure in hardware. In *Proceedings of the ISC Conference on Information Security and Cryptology (ISCISC'16)*.
[35] Amir Moradi, Bastian Richter, Tobias Schneider, and François-Xavier Standaert. 2018. Leakage detection with the x2-Test. *IACR Trans. Cryptog. Hardw. Embedd. Syst.* 1 (2018), 209–237.
[36] Adib Nahiyan, Mehdi Sadi, Rahul Vittal, Gustavo Contreras, Domenic Forte, and Mark Tehranipoor. 2017. Hardware Trojan detection through information flow security verification. In *Proceedings of the IEEE International Test Conference (ITC'17)*. IEEE, 1–10.
[37] Adib Nahiyan, Kan Xiao, Domenic Forte, and Mark Tehranipoor. 2017. Security rule check. In *Hardware IP Security and Trust*. Springer, 17–36.
[38] Jungmin Park. 2016. *Secure Hardware Design Against Side-channel Attacks*. Ph.D. Dissertation. Iowa State University.
[39] Gagandeep Singh. 2018. Gate-level Simulation Methodology. Retrieved from https://www.cadence.com.
[40] Céline Thuillet, Philippe Andouard, and Olivier Ly. 2009. A smart card power analysis simulator. In *Proceedings of the International Conference on Computational Science and Engineering (CSE'09)*, Vol. 2. IEEE, 847–852.
[41] Kris Tiri and Ingrid Verbauwhede. 2003. Securing encryption algorithms against DPA at the logic level: Next generation smart card technology. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 125–136.
[42] Nikita Veshchikov and Sylvain Guilley. 2017. Use of simulators for side-channel analysis. In *Proceedings of the IEEE European Symposium on Security and Privacy Workshops (EuroS&PW'17)*. IEEE, 104–112.
[43] Nicolas Veyrat-Charvillon and François-Xavier Standaert. 2009. Mutual information analysis: How, when and why? In *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems (CHES'09)*. Springer, 429–443.
[44] Kan Xiao, Adib Nahiyan, and Mark Tehranipoor. 2016. Security rule checking in IC design. *Computer* 49, 8 (2016), 54–61.
[45] Muhammad Yasin, Bodhisatwa Mazumdar, Sk Subidh Ali, and Ozgur Sinanoglu. 2015. Security analysis of logic encryption against the most effective side-channel attack: DPA. In *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS'15)*. IEEE, 97–102.

Received August 2019; revised February 2020; accepted February 2020