CISELEAKS: Information Leakage Assessment of Cryptographic Instruction Set Extension Prototypes

Aruna Jayasena, *Student Member, IEEE,* Richard Bachmann, *Student Member, IEEE,* and Prabhat Mishra, *Fellow, IEEE,*

***Abstract*—Software based cryptographic implementations provide flexibility but they face performance limitations. In contrast, hardware based cryptographic accelerators utilize application-specific customization to provide real-time security solutions. Cryptographic instruction-set extensions (CISE) combine the advantages of both hardware and software based solutions to provide higher performance combined with the flexibility of atomic-level cryptographic operations. While CISE is widely used to develop security solutions, side-channel analysis of CISE-based devices is in its infancy. Specifically, it is important to evaluate whether the power usage and electromagnetic emissions of CISE-based devices have any correlation with its internal operations, which an adversary can exploit to deduce cryptographic secrets. In this paper, we propose a test vector leakage assessment framework to evaluate the pre-silicon prototypes at the early stages of the design life-cycle. Specifically, we first identify functional units with the potential for leaking information through power side-channel signatures and then evaluate them on system prototypes by generating the necessary firmware to maximize the side-channel signature. Our experimental results on two RISC-V based cryptographic extensions, RISCV-CRYPTO and XCRYPTO, demonstrated that seven out of eight prototype AES- and SHA-related functional units are vulnerable to leaking cryptographic secrets through their power side-channel signature even in full system mode with a statistical significance of $\alpha = 0.05$.**

I. INTRODUCTION

In the modern landscape of information technology, cryptography, and its use cases have evolved into an essential tool for safeguarding sensitive data and ensuring secure communication. These requirements of cryptography extend far beyond its traditional role of encoding and decoding messages; they serve as a foundation for the confidentiality, integrity, and authenticity of digital information. From securing online transactions and protecting personal communications and digital privacy, cryptography plays a critical role in mitigating the ever-growing spectrum of cyber threats. As technology advances, there is an increasing demand for robust and fast cryptographic techniques, making it an integral component of our daily digital interactions.

Catering to these security demands, there are different techniques to implement cryptographic functionalities. The existing solutions can be mainly divided into three categories: software implementations, hardware accelerators, and cryptographic instruction set extensions. The distinction between software cryptographic implementations, cryptographic accelerators, and cryptographic instruction set extensions revolves around their specific approaches to managing cryptographic operations. Software cryptographic implementations, provided by libraries like OpenSSL [1], WolfSSL [2], Libgcrypt [3] and Crypto++ [4], utilize algorithms that are executed by the CPU through general-purpose instructions. However, their versatility may encounter performance limitations inherent in the nature of general-purpose processors. In contrast, cryptographic accelerators, such as Titan Security Key [5], IBM PCIe Cryptographic Coprocessor [6] and Trusted Platform Modules (TPMs) [7], [8], employ dedicated hardware components designed explicitly for cryptographic tasks, operating either independently or in parallel with the CPU to significantly boost processing power for cryptographic operations. Cryptographic instruction set extensions, like Intel Advanced Encryption Standard Instructions Set (AES-NI) [9], Intel SHA Extensions [10] and ARMv8-A Cryptography Extensions [11], strike a middle ground between software and hardware implementations using a hybrid approach by incorporating specialized instructions directly into the CPU architecture.

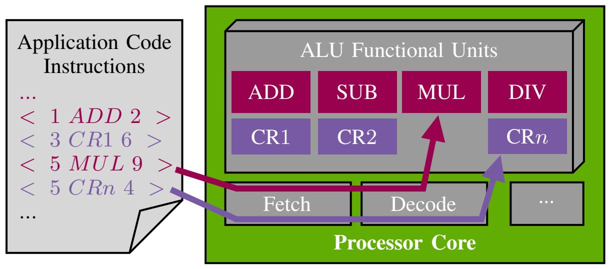


Fig. 1: Instead of using full cryptographic accelerators, a set of functional units (e.g., CR1) corresponding to cryptographic instruction set extensions are implemented in the arithmetic logic unit (ALU). This lets instructions related to cryptographic functions to be executed much faster and more efficiently compared to complete software implementations and more configurable compared to hardware cryptographic accelerators.

Figure 1 illustrates an abstract implementation concept of the Cryptographic Instruction Set Extensions (CISE). This approach aims to enhance performance without the need for separate hardware components for each cryptographic algorithm. Instead, each of the algorithm steps is divided into atomic cryptographic operations and each of these unique steps is implemented as a separate functional unit inside the processor. When it is required to perform a cryptographic workload, the pre-compiled software implementations consist of each of the specific cryptographic instructions that will be executed as atomic execution steps by the processor.

Whether a cryptographic function is implemented purely using software, hardware, or in a hybrid manner (CISE),

A. Jayasena, R. Bachmann and P. Mishra are with the Department of Computer & Information Science & Engineering, University of Florida, Gainesville, Florida, USA.

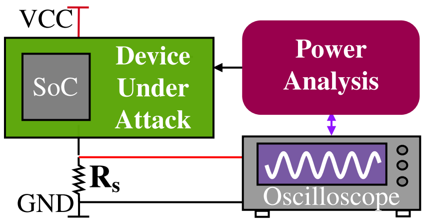


Fig. 2: Simple illustration of the setup for launching a power side-channel attack. An adversary can have a test setup that creates a power model of the device. In the field, the adversary can launch the attack on the victim device based on the model created using the test setup.

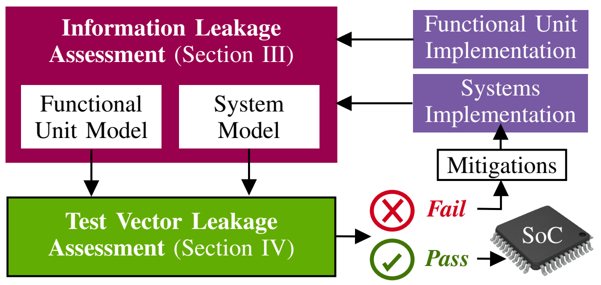

Information Leakage Assessment (Section III)
Functional Unit Model
System Model
Functional Unit Implementation
Systems Implementation
Mitigations
Test Vector Leakage Assessment (Section IV)
Fail
Pass
SoC

Fig. 3: Overall contribution of the proposed information leakage assessment framework. This consists of two main sections; unified information leakage assessment framework and test vector leakage assessment methodology.

they are susceptible to power side-channel attacks. Figure 2 illustrates a generalized setup that can be utilized by an adversary to mount an attack exploiting the power side-channel vulnerability. The adversary can obtain a test device with the same specification as the victim device and construct a model by manipulating the inputs and observing the power profile of the device. Then in the field, the adversary can mount the attack to recover the internal secret values that were leaked as a power side-channel signature from the victim device. Although software implementations can be masked with new software updates, the other two implementation techniques (hardware and CISE) are hard to mitigate if they are detected as vulnerable to power side-channel attacks after fabrication due to the inherent difficulty in modifying an integrated circuit.

This highlights the need for performing test vector leakage assessment of the hardware prototypes during the pre-silicon stage to detect potential power side-channel vulnerabilities in the early design life cycle. Although software-based masking techniques can be applied to the cryptographic instruction set extensions, such masking can add huge performance penalty defeating the purpose of having accelerated functional units to improve the performance. Therefore, similar to performing pre-silicon functional validation using simulation as well as formal verification, security validation of the cryptographic instruction set extensions using test vector leakage assessment is essential. Although test vector leakage assessment (TVLA) of cryptographic hardware has been explored in the literature [12]–[14], there are no prior efforts for evaluating cryptographic instruction set extensions that can perform TVLA of both the hardware and firmware components. In this paper, we propose an end-to-end pre-silicon test vector leakage assessment framework for cryptographic instruction set extension prototypes.

A. Research Contributions

In order to analyze cryptographic instruction set extension (CISE) prototypes, we propose a comprehensive framework, referred to as CISELEAKS, consisting of two evaluation rounds: 1) a functional unit evaluation round and 2) a full system evaluation with leaky functional units at early (pre-silicon) design stages. Each round will utilize a statistical test vector leakage assessment (TVLA) framework that assesses the potential power side-channel leakages. Figure 3 provides an overview of our proposed information leakage assessment framework. It accepts the hardware implementation and returns whether the given implementation can leak information as a power signature. Specifically, this paper makes the following contributions,

- We propose an input generation algorithm that can maximize the side-channel sensitivity of functional units.
- We formulate a methodology to evaluate prototype functional units for their power side-channel leakage.
- We propose a full system evaluation methodology to evaluate side-channel vulnerable functional units with the full system prototype.
- In support of system evaluation, we formulated an automated cryptographic workload generation mechanism to be used as the firmware for the system.
- We propose an automated trace alignment technique to detect the power consumption of the functional units from the full system power consumption.
- Evaluations on two RISC-V cryptographic instruction set extension (CISE) based designs, *RISCV-CRYPTO* [15] and *XCRYPTO* [16], have demonstrated that AES prototype implementations are vulnerable to leaking internal secrets as power side-channel signature.

The rest of the paper is structured as follows. First, we explore the background and related works in Section II. Next, we discuss the research contributions of the proposed approach in Section I-A. We elaborate on the major steps of the proposed information leakage assessment framework for cryptographic instruction set extensions in Section III and Section IV. In Section V, we apply the proposed technique to ongoing RISC-V cryptographic extension standardization work and show its effectiveness. Finally, we discuss the applicability and limitations of the proposed framework in Section VI and conclude the paper in Section VII.

II. Background and Related Work

In this section, we first discuss existing commercial and open-source implementations of cryptographic instruction-set extensions. Then, we examine how power side-channel attacks have been used for extracting cryptographic secrets from these implementations. Finally, we survey related efforts on pre-silicon test vector leakage assessment (TVLA) and discuss issues in using this technique to evaluate the implementations of cryptographic instruction set extensions.

A. Cryptographic Instruction Set Extensions

Several commercial implementations of scalar cryptographic extensions exist. An AES (Advanced Encryption Standard) extension, called AES-NI ("New Instructions"), was developed by Intel for the x86 instruction set architecture [17]. The extension includes instructions for encryption (AESENC), decryption (AESDEC), and key generation (AESKEYGENASSIST), with support for key sizes of 128, 192, and 256 bits. The first implementation of AES-NI has been developed by Intel for x86 architecture based processors and later similar functionality was adopted by AMD on several versions of their x86-based processors [18]. Another x86 extension, this time for SHA (Secure Hash Algorithm), was also developed by Intel [10]. It currently supports SHA-1 and SHA-256 and there are plans for supporting SHA-512 in the future. Implementations of this extension have been developed by both Intel (starting with the Westmere Sandy Bridge generation) and AMD (on their Zen and Puma processors). The ARMv8 instruction set architecture also features cryptographic extensions for both AES and SHA [19]. These extensions, denoted by the *+crypto* tag, have instructions for accelerating encryption and decryption (for AES), as well as for hashing operations (for SHA) [19].

There have also been efforts to develop cryptographic extensions for the open-source RISC-V instruction set architecture. According to a summary of RISC-V scalar cryptographic instruction set extensions [20], RISCV-CRYPTO [15] and XCRYPTO [16] are the two popular efforts [21]–[24]. Specifically, XCRYPTO [16] was developed considering the potential architectural side-channels that can be used by micro-architectural components [24]. Note that both RISCV-CRYPTO and XCRYPTO implementations are formally verified using *Sail* and *riscv-formal* verification frameworks separately. Both RISCV-CRYPTO and XCRYPTO implementations are open source. While the XCRYPTO extension has the complete prototype hardware implementations of the system with different functional units, RISCV-CRYPTO has hardware implementations of the functional units that can be used for the power side-channel evaluation.

B. Power Side-Channel and Cryptography

Power side-channel attacks present a considerable threat to cryptographic implementations. They work by monitoring variations in a device's power consumption in order to infer sensitive information. In the context of software implementations, these attacks target the power fluctuations caused by the execution of individual general-purpose instructions of the cryptographic algorithms. For instance, an adversary can analyze power consumption patterns during the execution of certain instructions, such as those involved in modular exponentiation in public key encryption. Software libraries like OpenSSL have been shown vulnerable to power side-channel attacks [25]. By monitoring power consumption during cryptographic operations, attackers can deduce secret keys, compromising the security of encrypted communications.

A power side-channel attack on the AES-NI extension is illustrated in [26]. Here, the authors were able to recover AES-NI keys from both an *SGX* enclave and the Linux kernel within a time frame of 26 hours. Variations in power consumption can also affect the electromagnetic characteristics of a device. This property was used to develop a side-channel attack against the *Apple iPhone 7* [27], a device featuring an ARM processor with cryptographic instruction set extensions. Specifically, the authors were able to successfully launch an attack on ARM/AES-CE implementation that utilizes the *ARMv8-A+crypto* extension, and they launched the side-channel attack focused on Apple's implementation of the specific instruction set on the *Apple A10 Fusion* System-on-Chip (SoC).

Cordwell et al. [28] performed a theoretical analysis of launching potential power side-channel attacks to reveal the initial seed input on SHA-2 family algorithms including SHA-512 using the Hamming weight of the input messages. The authors demonstrated the possibility of this attack using entropy/information theory arguments. The success of this attack is influenced by the word size used in the hash algorithm's operations; smaller word sizes make the side-channel attack more likely to succeed. If the algorithm happens to process input byte-by-byte, the attack is feasible. However, an algorithm that processes information in 64-bit words, as in SHA-512 and SHA-384, poses a much greater challenge to the adversaries. The effectiveness of this side-channel attack depends on the analyst's ability to measure near-perfect Hamming weights, which may be achieved through repeated measurements of identical hash operations. The theoretical possibility of extracting information from later rounds, given 80 rounds of processing and 20 independent input words, adds extra complexity. Success in launching an attack using the findings of this study depends on specific implementation details and device characteristics. A similar attack that can be launched on HMAC-SHA-2 and differential power analysis was proposed in [29]. The authors have utilized the Hamming distance leakage model on both pure hardware implementations on FPGA and software implementations to successfully launch an attack with less than 30K power traces.

It is important to highlight that the above vulnerable implementations were identified after the fabrication process of the hardware. Therefore, mitigation to prevent the leakage of the manufactured hardware adds huge performance penalties (e.g., with firmware-based masking techniques such as adding random instructions processing random data in between the actual cryptographic operations). This highlights the need for validation mechanisms at the early stages of the design life cycle of the cryptographic instruction set extensions.

C. Test Vector Leakage Assessment

Test Vector Leakage Assessment (TVLA) for hardware implementations aims to provide the following statistical assurance [30], [31]: the execution of the implementation doesn't directly or indirectly expose sensitive information through power side-channel signatures. There are promising TVLA techniques for hardware implementations of cryptographic implementations [12]–[14], [32]. Figure 4 illustrates the abstract steps involved in the pre-silicon test vector leakage assessment process for cryptographic implementations. The initial step involves generating tests based on Hamming distance to induce variations in power signatures [13], [14], [33]. Subsequently, the design undergoes simulation with the generated key pairs and a constant plaintext. The power signature is then derived from the change in values during the simulation. The disparity between two power signatures is computed using statistical techniques like *t-test and KL-divergence* [13], [14], [32]. Finally, the implementation is classified as either safe or susceptible to side-channel attacks based on a predetermined threshold. The same concept is applicable for public key cryptosystems with a few changes, such as stage-wise test vector leakage assessment on the vulnerable components and performing leakage assessment on sequential operations rather than block-wise operations involved in symmetric key cryptosystems [12].

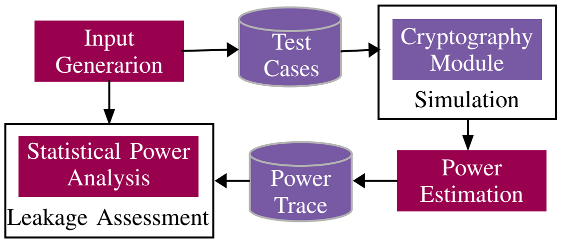


Fig. 4: An overview of pre-silicon test vector leakage assessment methodology [31] that consists of four majors steps. The first step is an input generation mechanism to maximize the side-channel sensitivity. Next, the designs are simulated with the generated inputs. Then, it generates a power consumption model for the device. Finally, it performs statistical evaluations to perform leakage assessment.

The main limitation of applying existing test vector leakage assessment (TVLA) techniques on the cryptographic instruction set extension (CISE) prototypes is due to the hybrid nature of the implementation which utilizes both hardware and firmware. Unlike software- and hardware-based approaches, CISE implementations are dependent on a special set of instructions on the firmware and how the compiler optimizes them. Moreover, each CISE instruction is executed using a custom functional unit that will have a unique power signature which needs to be evaluated.

III. Leakage Assessment of Cryptographic Instruction-Set Extension (CISE) Prototypes

In this section, we discuss the proposed information leakage assessment framework for cryptographic instruction set extension prototypes. Figure 5 illustrates the four major steps involved in the process. The first step is to identify the victim components of an implementation. This involves going through different cryptographic implementations to identify their vulnerable steps, such as collisions. Next, the functional units that implement these vulnerable components are evaluated for their potential information leakage using the test vector leakage assessment methodology. Then each of the functional units that fail the leakdown test are evaluated with the system again using the test vector leakage assessment methodology. Finally, if the system implementation passes the leakdown test for each of the functional units, then the implementation is ready for manufacturing. Otherwise, hardware mitigations should be applied to the system to mask the internal computations. The following subsections describe this process in detail with examples using evaluations on RISC-V *XCRYPTO instruction set extension*.

A. Victim Algorithm Identification

Before performing information leakage analysis, we have to identify whether a cryptographic algorithm is susceptible to power side-channel attacks, specifically those arising from cryptographic collisions. This process involves a literature review and a theoretical analysis. In this work, we survey the literature published by various international, national, and industry-specific cryptographic standards regulatory bodies, such as the National Institute of Standards and Technology (NIST), European Telecommunications Standards Institute (ETSI), Internet Engineering Task Force (IETF), and scientific research bodies, such as Office of Scientific and Technical Information (OSTI). Next, we utilize the information about research efforts on existing attacks on cryptographic implementations. After identifying such vulnerable algorithms, functional units corresponding to those algorithms should be considered for the next step of functional unit evaluation.

Example 1 (Vulnerable Algorithms): *In the case of the XCRYPTO instruction set extension, it supports AES and SHA cryptographic algorithms. There are theoretical as well as practical power side-channel attacks on both AES and SHA implementations on existing literature [9], [27]–[29], [34] as we discussed in Section II-B. Therefore, functional units corresponding to the parts of the implementations of AES and SHA should be considered for the functional unit evaluation.* ■

B. Functional Unit Evaluation Round

Once potential components that can leak sensitive information as power side channels are identified, the corresponding functional units need to be evaluated using test vector leakage assessment. Usually, all the functional units implemented inside the extension follow a certain standard of how they handle inputs and outputs. In addition to the input data registers, there are control flags that allow communication between the functional unit and the system. Therefore, a generic testbench can be used to evaluate all the functional units. For this, we create a testbench template that handles the control flags (such as $valid$ and $done$) and sets them to necessary input values. This testbench template is also responsible for feeding the controlled input data into the functional unit to perform the functional operations while dumping the simulation traces as a value change dump (VCD). Next, the hardware model is constructed by combining the hardware description of the functional unit (such as *Verilog implementation*) with the testbench (such as Verilator CPP testbench) into one compiled simulator application. This application can be sent to the test vector leakage assessment methodology (which will be discussed in Section IV) that will evaluate the functional unit for the information leakage via power side-channel signature. This methodology will evaluate the hardware model and return a "Pass" or "Fail" value for the side channel leakage. Here,

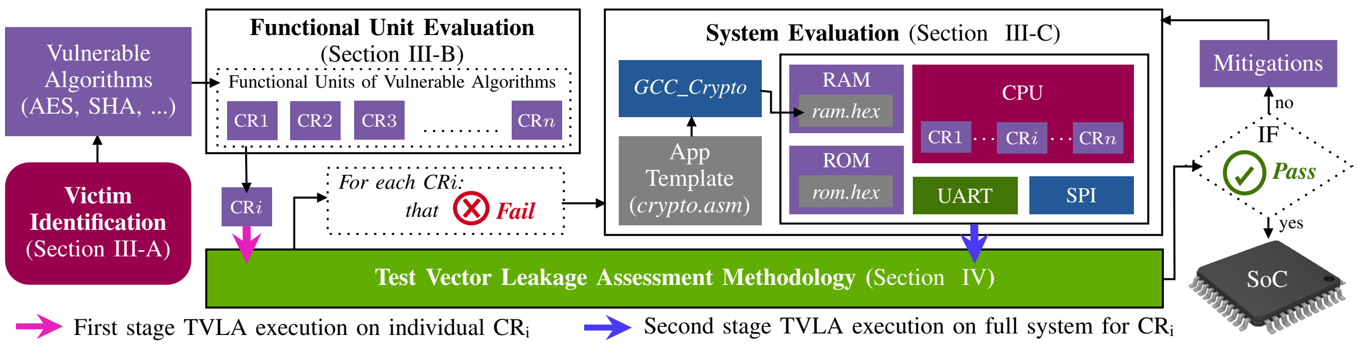

Vulnerable Algorithms (AES, SHA, ...)
Victim Identification (Section III-A)
Functional Unit Evaluation (Section III-B)
Functional Units of Vulnerable Algorithms
CR1
CR2
CR3
CRn
CRi
For each CRi: that Fail
System Evaluation (Section III-C)
GCC_Crypto
App Template (crypto.asm)
RAM
ram.hex
ROM
rom.hex
CPU
CR1 ··· CRi ··· CRn
UART
SPI
Mitigations
no
IF
Pass
yes
SoC
Test Vector Leakage Assessment Methodology (Section IV)
First stage TVLA execution on individual CRi
Second stage TVLA execution on full system for CRi

Fig. 5: Overview of information leakage evaluation framework for cryptographic instruction set extension prototypes. First, we identify the victim functional units that can leak sensitive information as a power signature. Next, we have two main evaluation rounds: individual functional unit evaluation (→) and full system evaluation (→). Each round prepares a corresponding hardware model and feeds that into the test vector leakage assessment methodology (Section IV). All the victim functional units are evaluated in the functional unit evaluation round and each of the failed functional units that is returned by the TVLA methodology are evaluated with the system evaluation round. Functional units that fail during the system evaluation should be mitigated with modifications to the prototype implementation.

the "Pass" signifies that the hardware model of the functional unit does not correlate with the input values indicating a power side-channel resistant implementation, and the "Fail" represents that the functional unit itself leaks input data as the side-channel signature and needs to be evaluated with the system.

Example 2 (Functional Unit Evaluation): *In case of XCRYPTO instruction set extension, we have evaluated four functional units of: xc_aesmix, xc_aessub, xc_sha*256*, and xc_sha*512*. All these functional units were classified by the test vector leakage assessment methodology as "Fail", indicating that they need to be evaluated with the system.* ■

C. System Evaluation Round

Once all the potential victim functional units are evaluated, all the "Failed" functional units need to be evaluated with the system. The reason behind the system evaluation is that, if other computations of the system can mask the operations of any functional unit, it will provide side-channel resistance to attacks against the particular functional unit. Compared to the functional unit evaluation, system evaluation is a complex process since it requires complete firmware binaries that can be simulated with the hardware implementation of the system. Further, this step requires the continuous integration and continuous delivery/continuous deployment (CICD) version of the *GNU Compiler Collection (GCC)* toolchain with the support for the cryptographic instructions to compile the binary. Next, we create a firmware template in assembly code that can switch between cryptographic workloads based on the context dynamically.

Figure 6 illustrates a firmware template that can be used for this purpose where `value_1` and `value_2` are inputs to the cryptographic functional unit that is under test. Next, in order to simulate the system, we construct a testbench that can read the compiled firmware as a *hex* file and feed it to the read-only memory (ROM) of the SoC while dumping the simulation trace as a value change dump (VCD). This firmware template is capable of initializing the CPU of the SoC into the proper state and writing the internal registers with the inputs to the cryptographic functional unit. Once the input data is loaded, the firmware makes a jump into the cryptographic workload. During the simulation process of the hardware model in the test vector leakage assessment methodology, the firmware will get updated with the input values of `value_1` and `value_2`. Note that each functional unit that needs to be evaluated with the system requires a separate workload with related cryptographic instructions.

```
.text
.global _start
_start:
    li x1 , 0
    li x2 , 0
    ...
    li x31, 0
    j  main
```

(a) System Initialization Function

```
.data
    value_1: .word 0x827b6f
    value_2: .word 0x1c42bff
main:
    la a1, value_1
    la a2, value_2
    nop
    j work
```

(b) Main Function

```
aes:
  xc.aessub.enc      a0,a1,a2
  xc.aessub.encrot   a0,a1,a2
  xc.aessub.dec      a0,a1,a2
  xc.aessub.decrot   a0,a1,a2
  xc.aesmix.enc      a0,a1,a2
  xc.aesmix.dec      a0,a1,a2
```

(c) AES Workload Template

```
sha:
  xc.sha256.s0  a0, a1
  xc.sha256.s1  a0, a1
  xc.sha256.s2  a0, a1
  xc.sha256.s3  a0, a1
```

(d) SHA Workload Template

Fig. 6: Cryptographic workload templates used for *XCRYPTO* instruction set extension running on *SCARV_SOC* system implementation. Depending on the functional unit that is under test, the assembly instruction '`j work`' should be changed to jump to the corresponding workload.

Example 3 (Workload Templates): *Figure 6 illustrates the abstract firmware template used for the evaluation of the XCRYPTO instruction set extension prototypes. Here the boot-loader code responsible for initializing the SoC properly is illustrated by Listing 6a and main function that writes input values to the internal registers is illustrated by Listing 6b.*

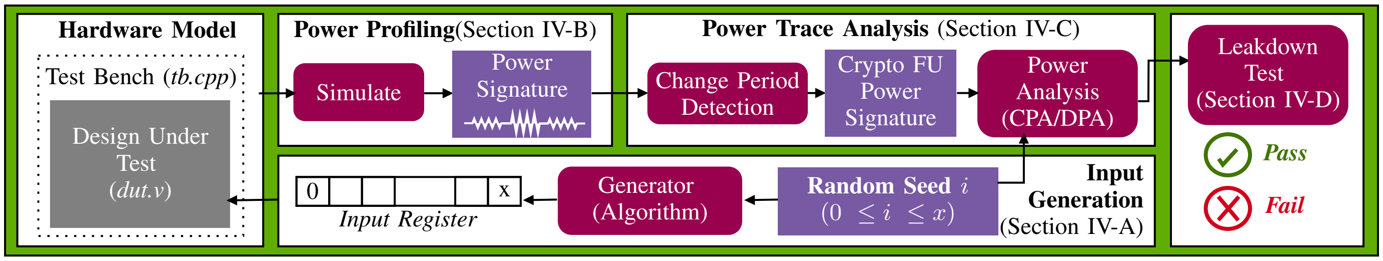

Hardware Model
Test Bench (tb.cpp)
Design Under Test (dut.v)
Power Profiling(Section IV-B)
Simulate
Power Signature
Power Trace Analysis (Section IV-C)
Change Period Detection
Crypto FU Power Signature
Power Analysis (CPA/DPA)
Leakdown Test (Section IV-D)
Pass
Fail
0
x
Input Register
Generator (Algorithm)
Random Seed i (0 ≤ i ≤ x)
Input Generation (Section IV-A)

Fig. 7: Overview of test vector leakage assessment that is used to evaluate individual functional units and full system of cryptographic instruction set extensions. The input to this methodology is the hardware model that includes a testbench with the implementation under test. This assessment consists of three major steps: input value generation, power profiling, and power trace analysis. A test called the leakdown test is performed to classify the hardware model as "Pass" (does not leak sensitive information as power side-channel signature) or "Fail" (leaks sensitive information as power side-channel).

In the case of the XCRYPTO, we need to evaluate both AES and SHA. Therefore, we have created two cryptographic workloads; the AES workload template illustrated in Listing 6c for evaluating the AES functional unit with the system and the SHA workload template illustrated in Listing 6d for the evaluation of SHA implementations with the system. During the compilation of the workload, the assembly instruction '`j work`' in Listing 6b is changed to jump to the corresponding workload that is under test. ■

Next, the system hardware model of the SoC with the firmware and the test bench is provided for the test vector leakage assessment methodology in Section IV which will return a "pass" or "fail" based on the statistical evaluations. If for all the functional units the system implementation "Passes" the evaluation, the instruction set extension prototype implementation passes the test vector leakage assessment and the SoC is ready for the manufacturing process. However, if at least one of the functional units "Fails" the system evaluation, modifications are needed to mitigate the power side-channel leakage. After applying the mitigation, the same experiment should be repeated until it does not leak information as a power side-channel signature.

IV. Test Vector Leakage Assessment

In the previous section, we have discussed the steps involved in transforming a pre-silicon design for evaluation. In this section, we discuss the specific steps involved in the test vector leakage assessment. Figure 7 provides an overview of the proposed test vector leakage assessment methodology. First, we generate inputs to be fed into the cryptographic workloads. Next, we simulate the implementation and obtain the power signature. Then, we perform trace analysis to quantify the amount of information leakage. Finally, we perform a leakdown test to return a "Pass" or "Fail" result on the evaluation.

A. Input Generation

The idea of this step is to manipulate the inputs to the hardware implementation to maximize the side-channel sensitivity. This facilitates the evaluation mechanism to observe the power fluctuations and correlate the inputs with the observed power fluctuations. Transistors, as fundamental building blocks of hardware circuits, determine the power consumption of the underlying implementation. The Hamming Weight Model and Switching Activity Model are two approaches commonly employed for estimating the power of the hardware design.

- *Hamming Weight (HW) Component (v_{hw})*: The power consumed by a register or a memory element is proportional to the number of bits set to '1'.
- *Switching Activity Component (v_{sw})*: The power consumption is related to the switching activity in the logic gates and interconnects of the design.

Therefore, in order to improve the side-channel sensitivity of an implementation, the Hamming weight of the inputs needs to be manipulated in a way that they follow a uniform distribution. For this purpose, we utilize a modified version of "*Algorithm L*" [35] which is used for Lexicographic Permutation Generation. The steps of the modified algorithm are illustrated in Algorithm 1. This algorithm generates a random number of a given Hamming weight. In order to generate sequences of random numbers for manipulating the implementation, we randomly sample the Hamming weight for each of the inputs from a uniform distribution. Let's assume the register architecture of the instruction set extension is X. Then the input sequence that an implementation under test will be simulated with can be represented as shown in Equation 1.

$$\{hwGen(r_i) \mid r_i \in [0, X), i \in \mathbb{N}\} \tag{1}$$

Algorithm 1 Input Generation using $hwGen()$ function

Input: Hamming weight hw, generator g, register width w
Output: random number R

```
1: function hwGen(hw, g, w)
2:     $R \leftarrow 0$
3:     for $i \leftarrow 0$ to hw $- 1$ do
4:         bitPosition $\leftarrow U(0, w-1-i)(g)$
5:         $R \leftarrow R \mid (1 \ll$ bitPosition$)$
6:     end for
7:     return R
8: end function
```

Algorithm 1 essentially generates random inputs with uniformly distributed Hamming weights that can be fed into the functional unit of the implementation under test. This effect cannot be obtained by directly using randomly sampled inputs since random numbers do not have uniformly distributed randomness among their Hamming weights. Figure 8 illustrates the Hamming weight of the numbers generated using the function $hwGen()$ of Algorithm 1 (🟦 `hw(hwGen)`) compared with the Hamming weights of uniformly sampled random numbers (🟥 `hw(random)`). It can be observed that the Hamming weights of the inputs generated by the function $hwGen()$ are evenly distributed in the input space compared to the Hamming weights of the uniformly sampled random numbers. A weighted combination of the effect of both Hamming weight and the switching activity accounts for the input-based expected power consumption (v^f) of the functional unit, as illustrated in Equation 2.

$$v^f = w_1.v_{hw} + w_2.v_{sw} \tag{2}$$

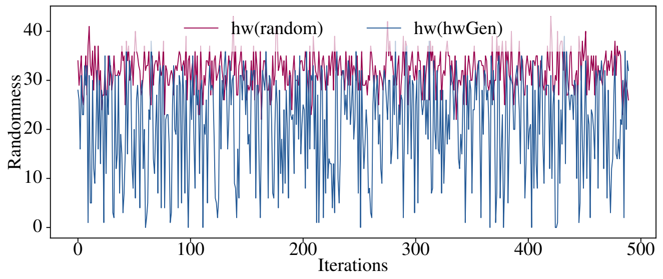


Fig. 8: Comparison between the randomness of the Hamming weights generated using the function $hwGen$ of Algorithm 1 (🟦 `hw(hwGen)`) against the Hamming weights of uniformly sampled random numbers (🟥 `hw(random)`).

However, power variations induced by Hamming weights and switching activity are only correlated to the operation of individual functional units. This is similar to the approach followed by the traditional method of TVLA which assumes a more isolated execution environment. When it comes to the system evaluation stage of the CISE, we need to account for complex interactions between the firmware, hardware, and the operating system kernels. As a result, applying traditional TVLA methods directly to CISE could miss critical leakage paths or incorrectly attribute leakage to irrelevant operations, leading to inaccurate assessments of the system's side-channel resistance. For this, we incorporate two additional components into the power model of the system evaluation round.

- *Flag Interaction Component* (v_{fi}): Cryptographic ISAs may affect CPU flags (e.g., zero flag, carry flag) during the execution of cryptographic instructions, which could contribute to the overall power consumption.
- *Instruction-based Component* (v_{in}): Each CISE instruction has a base power consumption based on its complexity, number of cycles, and hardware resources used.

Finally, we use the weighted sum of these components to compute the input-based expected power consumption v^s of the system as illustrated in Equation 3.

$$v^s = w_1.v_{hw} + w_2.v_{sw} + w_3.v_{fi} + w_4.v_{in} \tag{3}$$

Computing each of the weights can be done with the help of post-synthesis results. Post-synthesis reports provide details, such as net capacitances, gate types, and toggle rates, which serve as inputs to compute dynamic power. By applying the technology library's power models, an initial estimate can be computed. The scaling factor is then adjusted based on realistic conditions, accounting for factors like wire loads, parasitics, and temperature, ensuring a closer approximation to post-silicon measurements. Due to the scope of this work and the nature of the reference implementation, we use an average-case estimation by using equal weights for each component.

B. Power Profiling

Once input patterns are generated, the next step is to feed the generated inputs into the hardware implementation and simulate it. The steps involved in the testbench development process for functional units and system evaluation were discussed in Section III-B and Section III-C, respectively. Note that the hardware model that is provided as the input to this step is a compiled simulator program that accepts sequences of 1) register input values in the case of functional unit evaluation and 2) compiled firmware in the case of full system evaluation. Therefore, for the functional unit evaluation, the input values generated in Section IV-A are directly provided as inputs while for the system evaluation, the firmware needs to be updated with the input values generated in Section IV-A and compiled. The next step is to obtain the estimate of the power consumption of the fabricated chip using the pre-silicon simulation. To accomplish this, we use simulation value change dump (VCD) traces and estimate the power consumption of the transistors in the design with power estimation tools.

The VCD captures signal activity during simulation, including transitions of nets and registers. These transitions are used to compute dynamic and static power consumption. The typical flow involves running a functional simulation with the generated inputs to obtain the VCD file, which is then analyzed considering both the switching activity and the design's capacitance model. This process helps estimate power consumption based on real signal toggling in the design [12], [13], [32], [33], [36]–[38]. There are commercial tools such as Synopsys PrimePower [39], Cadence Voltus [40], and Siemens PowerPro [41] that can provide detailed power analysis using VCD files. These tools offer advanced features like clock gating, glitch filtering, and hierarchical analysis, making them suitable for both early-stage and more detailed power estimations.

C. Power Trace Analysis

Once the power signature of the implementation is extracted from the simulation, the specific region of the power signature that is responsible for the cryptographic functional unit needs to be isolated. We first perform a change period detection. Once the power signature is isolated, correlation power analysis and differential power analysis can be performed, which will evaluate the correlation between the power signature and input values to the cryptographic functional units. In this section, we discuss these three steps in detail.

*1) **Change Period Detection:*** In order to extract the power signature corresponding to the functional unit from the entire simulation, we first need to align the input sequences with the power traces. This process is done based on the pipeline depth of the design under test. For example, in case of a combinational functional unit that implements the dataflow behavior, the power signature is the power values observed in the next cycle right after feeding the inputs. On the other hand, in case of a sequential functional unit that consumes a fixed number of cycles to complete the operations, the corresponding power values are found in a region, which we refer as "Change Period (C_p)". The change period can be visually identified in the power trace by changing only the inputs of the functional unit and keeping all other inputs in fixed values.

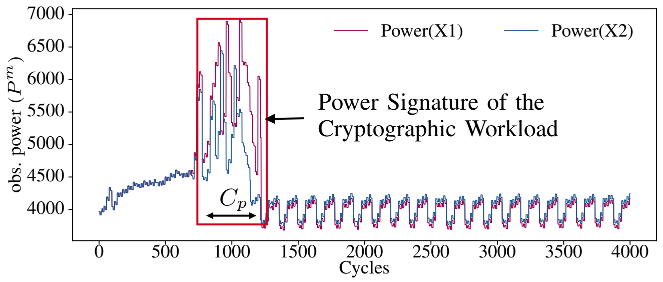


Fig. 9: Change period (C_p) detection from the full system (executing cryptographic workload AES firmware on SCARV-SOC) power signature evaluation. By only changing the input register values from $X1$ to $X2$ for the functional unit on the particular firmware, the power consumption period of the functional unit can be uniquely identified.

Example 4 (Change Period Detection): *Figure 9 illustrates an example where we have used the proposed change period detection technique on an AES functional unit of the XCRYPTO extension with the SCARV-SoC. In this instance, the firmware is unchanged except for the fact that the input values to the AES functional unit are changed from $X1$ to $X2$. This drastically changes the power signature of the functional unit, which makes it distinguishable from the power signature created by other components of the SoC.* ■

Once the C_p is identified, we map it to a single power value (p^m) by considering the maximum observed power point within the range of C_p as illustrated in Equation 4. The reason for this is that an adversary is interested in the peak power points in the power signature since that is observable during the actual device is in the field.

$$p^m = max(C_p = \{p_1, \ldots, p_j\}) \tag{4}$$

Next, we need to determine the minimum number of experiments (n) that need to be performed in order to achieve the required statistical significance level of α. In order to calculate this, we repeat the above process with 1000 experiments and collect the peak power distribution that contains 1000 samples as $P^{1000} = \{p_1^m, \ldots, p_{1000}^m\}$. Next, we use Equation 5 to compute the n value using the collected peak power distribution. Here, Z_i is the point on the normal distribution to give the required statistical power and significance. Additionally, d represents the effect size, σ is the standard deviation, while β and α represent the statistical power and significance respectively. Next, the value of the n is obtained and the change period detection experiment is repeated until it satisfies the required n peak power samples.

$$n = 2 \cdot \left(\frac{Z_{(1-\frac{\alpha}{2})} + Z_\beta}{d}\right)^2 \cdot \sigma^2 \tag{5}$$

Once this process is repeated for the minimum number of experiments (n) required for the required statistical significance, we can obtain a distribution that consists of the peak power of each experiment as $P^m = \{p_1^m, \ldots, p_n^m\}$ which will be used to perform the correlation power analysis in the next step. This will result in distribution with either $V^f = \{v_1^f, \ldots, v_n^f\}$ or $V^s = \{v_1^s, \ldots, v_n^s\}$ representing input-based expected power consumption depending on the corresponding evaluation round of functional unit or system evaluations, respectively. For ease of reference, we will refer to this distribution as $V^{f/s}$ throughout the rest of this section. This essentially refers to V^f during the functional unit evaluations and V^s during the system evaluation round.

*2) **Correlation Power Analysis:*** At this stage, we have two distributions of expected power consumption values ($V^{f/s}$ for functional units or the system) generated by Algorithm 1 and peak power (P^m) obtained in Section IV-C1 which contains n samples in each. Lets represent these two distributions as $V^{f/s} = \{v_1, \ldots, v_n\}$ and $P^m = \{p_1^m, \ldots, p_n^m\}$, respectively. For the correlation power analysis, we will be conducting hypothesis testing. Therefore, we construct the hypotheses

- H_0 as *there is no correlation between the observed power consumption against the input-based expected power consumption*
- H_1 as *there is a correlation between the observed power consumption and the input-based expected power consumption*

Then we set the statistical significance to α which is used to calculate the sample size (n). Next, we compute the Chi-squared static for two distributions using Equation 6. Here, p_i corresponds to each element in the peak power distribution P^m. The expected power value e_i is calculated using Equation 7 and the contingency table that is constructed using both the distributions of $V^{f/s}$ and P^m. In Equation 7, γ, ν, Λ represent row sum, column sum, and the total sum of the contingency table, respectively.

$$\chi^2 = \sum_{i=1}^{n} \frac{(p_i - e_i)^2}{e_i} \tag{6}$$

$$E_i = \frac{\gamma(W(v_i)) \times \nu(p_i)}{\Lambda} \tag{7}$$

A contingency table is constructed to organize and summarize the joint distribution of two categorical variables of peak power distribution and the expected power consumption based on the input. To create a contingency table, we assign each variable either a row or a column, and the intersection cells represent the frequency of corresponding observations falling into that category. In other words, the contingency table can translate the expected power consumption based on the input into a corresponding peak power value.

		Observed Peak Power Values (P^m)									
		p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
Exp. Power (V^f)	v_1	3	8	5	2	7	4	10	6	9	1
	v_2	7	1	9	6	3	8	2	10	4	5
	v_3	1	4	2	9	8	3	6	7	10	5
	v_4	5	6	8	3	1	10	7	9	2	4
	v_5	8	3	4	7	5	2	1	6	10	9
	v_6	2	7	1	10	9	5	4	8	3	6
	v_7	9	10	6	4	2	9	3	1	7	8

Fig. 11: Example contingency table constructed from input-based expected power consumption value of the functional units with the observed peak power values.

Example 5 (Contingency Table): *Figure 11 illustrates an example contingency table constructed from two example peak power and input-based expected power consumption distribution. In this example, we have observed ten unique power levels in the peak power consumption distribution from p_1 to p_{10} which is represented as each column. On the input-based expected power consumption distribution, we have observed seven different values from $v1$ to $v7$ which are represented in each row. Then the internal cell values are the frequencies of both occurrences at the same time. For example, the entry corresponding to the first row (v_1) and the first column (p_1) indicates that we have observed 3 samples with the input-based expected power consumption of v_1 and the peak power value of p_1.* ■

$$df = (|\gamma| - 1).(|\nu| - 1) \tag{8}$$

$$p\text{−}value = 1 - CDF(\chi^2, df) \tag{9}$$

Next, we use Equation 8 to compute the degree of freedom df, where $|\gamma|$ represents the number of rows and $|\nu|$ represents the number of columns in the contingency table. For the contingency table in Figure 11, the degree of freedom is $df = (7 - 1) \times (10 - 1) = 54$. Finally, using the cumulative distribution function (CDF) with the computed χ^2 value from Equation 6 and df from Equation 8, the $p\text{−}value$ is computed using Equation 9. If $p\text{−}value \leq \alpha$, we reject the null hypothesis (H_0), which indicates that there is a significant correlation between the peak power consumption of the implementation with the inputs to the cryptographic function units. Alternatively, if $p\text{−}value > \alpha$, we fail to reject the null hypothesis (H_0).

3) ***Differential Power Analysis****:* Differential Power Analysis (DPA) enhances Correlation Power Analysis (CPA) by grouping power traces based on key guesses and analyzing the differences in power consumption, which enables efficient detection of key-dependent leakage, even in noisy environments [42]. DPA can further improve the analysis by incorporating higher-order power analysis [43], which targets more complex leakage scenarios involving multiple intermediate values. Specifically, we use DPA in the system-level analysis, where noise can be introduced by other components, such as the processor pipeline and the compiled code (assembly instructions executed in the pipeline). Similar to the CPA, we conduct hypothesis testing for the DPA. Therefore, we construct the hypotheses

- H_0 as *there is no correlation between the observed power consumption against the input-based expected power consumption*
- H_1 as *there is a correlation between the observed power consumption and the input-based expected power consumption*

Next, we select a possible guess for the algorithm's secret key. Then, for each input value in $V^{f/s}$, we compute the input-based expected power consumption considering the guessed key as the input. After that, we group the input values $V^{f/s}$ and corresponding power traces P^m into two groups, one where the hypothetical power matches the key guess and one where it does not. Then for each group, we calculate the average power trace over all the samples. Finally, we compute the difference between the average traces of the two groups to identify potential key-dependent leakage using t-statistic illustrated by Equation 10 (with a corresponding $p\text{-}value$ value in Equation 11). In Equation 10, μ_{g1} and μ_{g0} corresponds to means of the two groups and σ_{g1} and σ_{g0} are the variances, and n_{g1} and n_{g0} are the sample size of two groups.

$$t = \frac{\mu_{g1} - \mu_{g0}}{\sqrt{\frac{\sigma_{g1}}{n_{g1}} + \frac{\sigma_{g0}}{n_{g0}}}} \quad (10) \qquad p\text{-}value = 2\int_{|t|}^{\infty} f(t,d)dt \quad (11)$$

We use the $p\text{-}value$ from Equation 11, where $f(t, d)$ is the probability density function of the t-distribution with d degrees of freedom. The null hypothesis H_0, which assumes no relationship between power consumption and the key guess, is rejected if the $p\text{-}value$ is below a chosen significance level α. If the $p\text{-}value < \alpha$, it indicates the presence of higher-order key-dependent leakage, rejecting H_0. Otherwise, H_0 is accepted.

D. Leakdown Test

Leakdown test determines whether each functional unit leaks internal secret information as a power side-channel signature based on the correlation power analysis results and differential power analysis results by assigning a "Pass" or "Fail" result. Here "Fail" result happens when we reject the null hypothesis ($p\text{−}value \leq \alpha$) for either of the CPA or DPA results which means that the functional unit leaks the information about the cryptographic secrets as power side-channel signature. In other words, in order to leakdown test to be passed, both CPA and DPA results should accept their corresponding null hypothesis H_0.

Since the proposed leakage assessment methodology is comprised of two rounds of functional unit evaluation (discussed in Section III-B) and system evaluation (discussed in Section III-C), the leakdown test is carried out as follows. First, each functional unit is evaluated with the test vector leakage assessment methodology in the functional unit evaluation round to obtain the leakdown test results. Next, all failed functional units are sent to the system evaluation round. The objective is that if other components in the system can mask the power signature of the functional unit, then the implementation will not leak the information. The leakdown test results for system evaluation round corresponding to each functional unit will classify the testing prototype implementation as a ”Pass” or ”Fail” from the test vector leakage assessment. Any functional units that ”Fails” the leakdown test should incorporate register masking techniques to hide/obfuscate the power signature and the modified implementation should be again evaluated with our proposed approach until it passes the system evaluation round.

V. Experiments

In this section, we evaluate two prototype implementations of cryptographic instruction set extensions for the open-source RISC-V architecture. We first briefly introduce these two instruction set extensions. Next, we outline our experimental setup. Finally, we present our experimental results.

A. Instruction Set Extensions Under Evaluation

In order to evaluate the effectiveness of proposed test vector leakage assessment framework, we have selected two popular cryptographic instruction set extensions, *RISCV-CRYPTO* and *XCRYPTO* . In this section, we provide a brief overview of these implementations.

RISCV-CRYPTO instruction set extension [15]: RISCV-CRYPTO extension was proposed as a lightweight accelerator solution for cryptographic workloads of embedded systems. Modern cryptographic operations work with operands wider than the individual elements in modern computer architecture, which are typically limited to 64 bits. These wider operands, often 128 or 256 bits, can consist of smaller elements that are combined or may be a single value (e.g., 128-bit block or round key in AES). RISCV-CRYPTO treat these operands as vectors of one or more element groups based on the RISC-V Vector Element Groups specification. Each vector crypto instruction explicitly defines three parameters, Element Group Width (EGW), Effective Element Width (EEW) and Element Group Size (EGS), which represents total number of bits in an element group, number of bits in each element, and the number of elements in an element group, respectively. Table I presents the specification details of different cryptographic algorithms with the three parameters of EGW, EEW, and EGS.

XCRYPTO instruction set extension [16]: XCRYPTO aims to facilitate efficient and secure software implementation of cryptographic primitives, similar to standard floating-point extensions. It explores a diverse design space for processor cores and system architectures, allowing for hardware-only, mixed, or firmware-only approaches. However, XCRYPTO specifically focuses on supporting firmware-based cryptographic implementations, with an emphasis on constrained cores like microcontrollers. The specification does not assume a specific value for architecture register widths (32, 64, or 128), but it commonly targets 32-bit microcontroller-class cores. XCRYPTO requires interaction with a Random Number Generator (RNG), leaving the instantiation unspecified but assuming adherence to best practices for security. This approach balances flexibility in implementation with the critical importance of selecting a secure RNG instance. Compared to RISCV-CRYPTO, XCRYPTO is in a more mature stage in its development. It consists of Verilog prototypes for each of the required functional units for cryptographic algorithms of AES, SHA256, and SHA512. It consists of a complete SoC implementation in Verilog with a CPU that integrates the prototype functional units enabling full pre-silicon system simulation.

TABLE I: Specification details of different cryptographic algorithms on RISCV-CRYPTO instruction set extension.

Algo.	**AES**	**SHA256**	**SHA512**	**GCM**	**SM4**	**SM3**
Extn.	Zvkned	zvknh	zvknhb	Zvkg	Zvksed	Zvksh
EGW	128	128	265	128	128	256
EEW	32	32	64	32	32	32
EGS	4	4	4	4	4	8

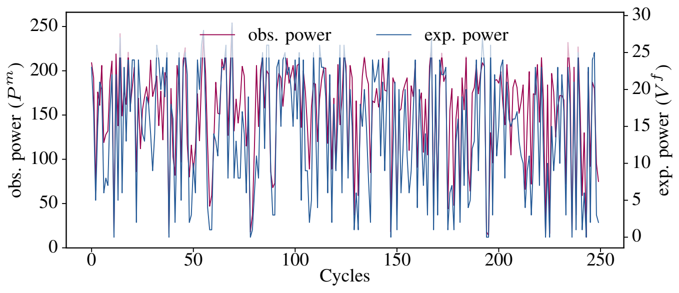
B. Experimental Setup

We have used Verilog prototype implementations of both RISCV-CRYPTO and XCRYPTO extension prototypes. For the systems evaluation of the XCRYPTO extensions, we have used SCARV-SOC with the SCARV-CPU that integrated the prototype functional units. Verilator [44] simulator was used to simulate the Verilog implementations and obtain the simulation traces as value change dumps (VCD). Pre-silicon power estimation based on VCD was performed with *Synopsys PrimePower [39]*. For compiling the firmware for the system evaluation, the modified GNU GCC toolchain with XCRYPTO extension was used. The process of building the system model, automated compilation of firmware and the power modeling was performed using C++ and C while statistical computations were performed using Python scripts. The entire framework was implemented inside a Docker environment and the experiments were performed on a system with an Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz (64 bit) with 24GiB memory. According to the analysis of the cryptographic algorithms and their associated collisions that can reduce the effort of key/secret guessing, we have selected the functional units corresponding to the algorithms SHA and AES.

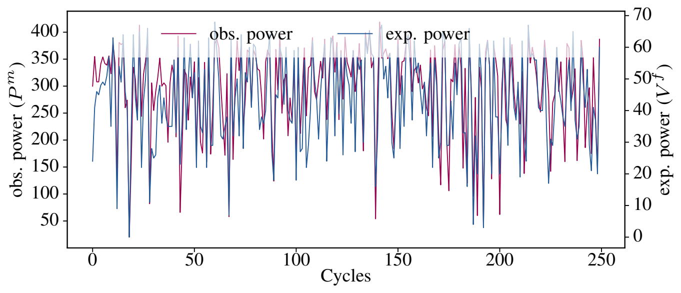
C. SHA Functional Unit Evaluation Results

In this section, we present the results for evaluating SHA functional units that are implemented in Verilog from RISCV-CRYPTO and XCRYPTO extensions. For this, we have simulated the prototype implementations according to the steps outlined in Section III-B. Table II presents the configuration parameters and the results of the experiments for evaluating SHA implementations. The statistical results in Leakdown test column illustrate that there is a strong correlation between the input-based expected power consumption and the observed power consumption of individual functional units, which leaks the cryptographic secrets as power side-channel signature.

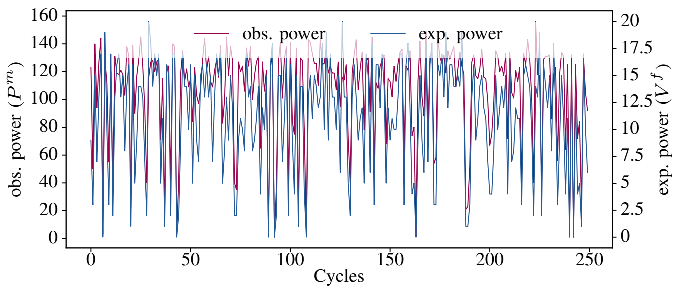
In order to visually observe this correlation we have plotted the observed power consumption against the input-based expected power values. Figure 13 demonstrates the visual similarity between the observed power values (■ `obs. power`) of the functional units against the input-based expected power consumption (■ `exp. power`) of the functional unit. Here Figure 13a, Figure 13b, Figure 13c, and Figure 13d represent the first 256 experiments out of all conducted



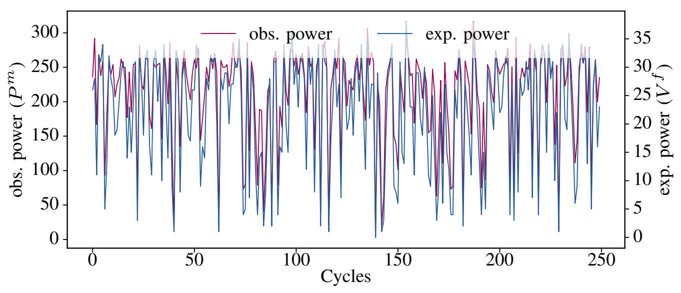
(a) RISCV-CRYPTO SHA256



(b) RISCV-CRYPTO SHA512



(c) XCRYPTO SHA256



(d) XCRYPTO SHA512

Fig. 13: Visual similarity between the observed peak power consumption (🟥 `obs. power`) of the functional unit and the input-based expected power values (🟦 `exp. power`) on functional modules related to SHA computations on RISCV-CRYPTO and XCRYPTO cryptographic instruction set extension prototypes. The variations in the trend of the input-based expected power values are preserved in the observed power signature.

TABLE II: Test vector leakage assessment results on SHA functional unit prototypes from RISCV-CRYPTO and XCRYPTO extensions. Here all the functional units fail the leakdown test indicating a high correlation between the input-based expected power consumption against the observed power consumption of the functional unit.

	RISCV-CRYPTO		*XCRYPTO*	
Module	*SSHA256*	*SSHA512*	*SHA256*	*SHA512*
Minimum n	4896	5103	4972	5001
Eval. Time (Sec)	6	5	5	6
CPA *p-value*	6.534e-7	7.548e-7	2.838e-7	8.991e-7
Leakdown Test	Fail	Fail	Fail	Fail

TABLE III: Test vector leakage assessment results on AES functional unit prototypes from RISCV-CRYPTO and XCRYPTO extensions. Here all the functional units fail the leakdown test indicating a high correlation between the input-based expected power consumption against the observed power consumption of the functional units.

	RISCV-CRYPTO		*XCRYPTO*	
Module	*SAES32*	*SAES64*	*AESMIX*	*AESSUB*
Minimum n	5293	5209	5013	5320
Eval. Time (Sec)	11	10	13	14
CPA *p-value*	6.892e-7	5.473e-7	8.857e-7	7.921e-7
Leakdown Test	Fail	Fail	Fail	Fail

experiments of four instances of experiments on RISCV-CRYPTO SHA256, RISCV-CRYPTO SHA512, XCRYPTO SHA256, and XCRYPTO SHA512, respectively. It illustrates that prototype implementations from both RISCV-CRYPTO and XCRYPTO have a high correlation between the input-based expected power with the observed power signature of the implementation.

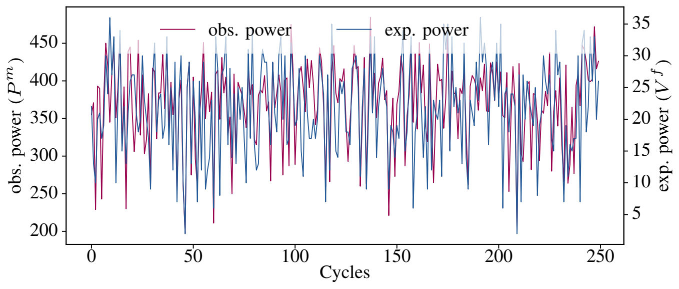
D. AES Functional Unit Evaluation Results

In order to evaluate the AES implementations of both RISCV-CRYPTO and XCRYPTO prototype extensions, we have applied the steps outlined in Section III-B on the corresponding Verilog functional units. Table III presents the configuration parameters and the results of the experiments for evaluating AES implementations. Similar to the SHA implementation results, the leakdown test results for AES functional modules illustrate that there is a strong correlation between input-based expected power consumption and the observed power consumption of individual functional units.

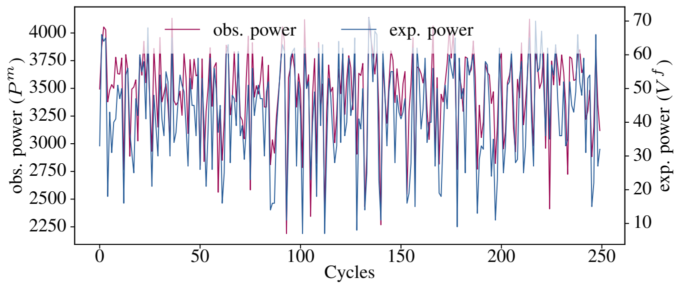
Figure 13 demonstrates the visual similarity between the observed power values (🟥 `obs. power`) of the functional units against input-based expected power consumption (🟦 `exp. power`) of the functional units. Here Figure 14a, Figure 14b, Figure 14c, and Figure 14d represent the first 256 experiments out of all the conducted experiments of four instances of experiments on RISCV-CRYPTO AES32, RISCV-CRYPTO AES64, XCRYPTO AESMIX, and XCRYPTO AESSUB, respectively. It illustrates that AES prototype implementations from both RISCV-CRYPTO and XCRYPTO have a high correlation between the input-based expected power consumption and the observed power signature of the implementation.

E. System Evaluation Results

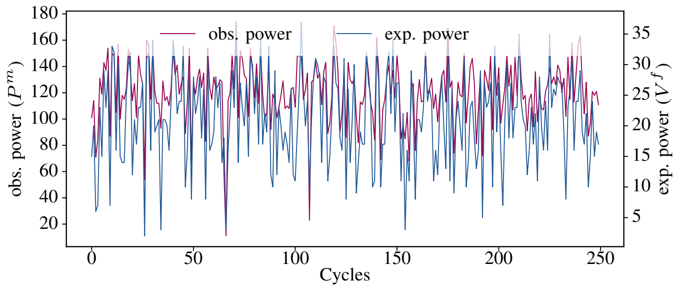
Since all the functional units related to the cryptographic algorithms of AES and SHA failed the functional unit evaluations signifying that there is a considerable information leakage, we have performed the system evaluation as discussed in Section III-C. However, since both RISCV-CRYPTO and



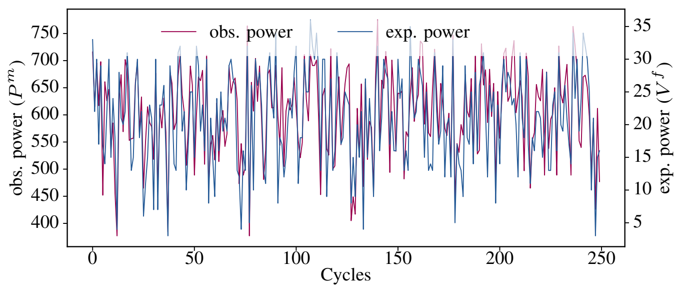
(a) RISCV-CRYPTO SAES32



(b) RISCV-CRYPTO SAES64



(c) XCRYPTO AESMIX



(d) XCRYPTO AESSUB

Fig. 14: Visual similarity between the observed peak power consumption (■ `obs. power`) of the functional unit and the input-based expected power values (■ `exp. power`) on functional modules related to AES computations on RISCV-CRYPTO and XCRYPTO cryptographic instruction set extension prototypes. The variations in the trend of the input-based expected power values are preserved in the observed power signature.

XCRYPTO are in the development phase, all the functional units are not integrated into full system prototypes. The full system prototype is available for only three functional units of SHA256, AESSUB, and AESMIX from the XCRYPTO instruction set extension. Table IV presents the configuration parameters and the results of the experiments for evaluating SHA256, AESSUB, and AESMIX implementations. The 'Fail' statistical results in the Leakdown test column illustrate that there is a strong correlation between the input-based expected power consumption and the observed power consumption of the system on both AESSUB and AESMIX prototypes, which leaks the cryptographic secrets as power side-channel signature. On the other hand, the SHA256 implementation 'Pass' the leakdown test signifying that it does not have a statistical correlation between the input-based expected power consumption and the observed power consumption of the system implementation.

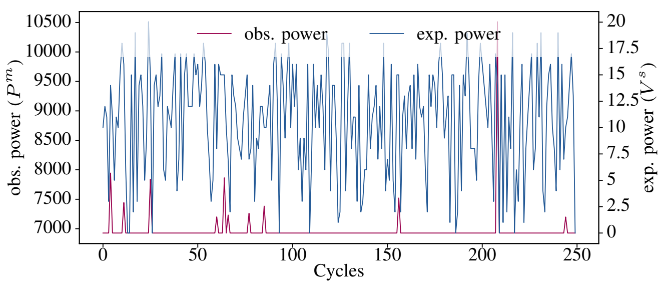
Similar to the previous experiments, we have plotted the visual relationship between the system power consumption against the input-based expected power consumption related to the system, when executing the firmware. Figure 15 demonstrates the visual similarity between the observed power values (■ `obs. power`) of the functional units against the input-based expected power consumption (■ `exp. power`) of the system. Here Figure 15a, Figure 15b, and Figure 15c represent the maximum power values observed in the first 256 experiments out of all the experiments of each power trace analysis round for three instances of XCRYPTO SHA256, XCRYPTO AESMIX, and XCRYPTO AESSUB, respectively. As expected, there is no visible correlation for SHA (Figure 15a) but a strong correlation for AES (Figure 15b and Figure 15c). Unlike functional unit evaluations, system evaluations include the noise added by the compiler and the processor pipeline such as the instruction fetch, decode, execute, etc. Additionally, memory transfers occur during the execution of the firmware. In the case of SHA, the effects from the above components hindered the power side channel leakage that we observed on the isolated functional unit.

TABLE IV: Test vector leakage assessment results on full system prototypes of XCRYPTO extensions. Here the two functional units related to AES computations fail the leakdown test indicating a high correlation between the input-based expected power consumption against the observed power consumption of the system. However, SHA computation passes at the system level, which indicates that other system operations have masked the unit level leakage (shown in Table II).

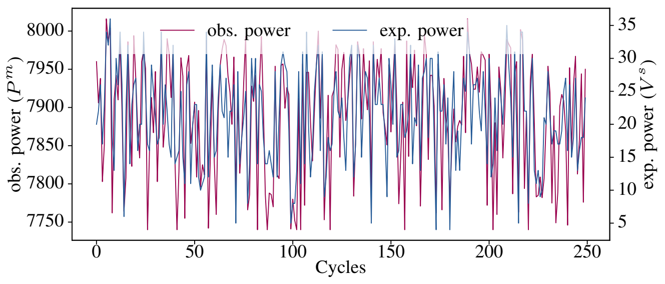
SoC (Extension)	*SCARV-SoC (XCRYPTO)*		
Module	*SHA256*	*AESMIX*	*AESSUB*
Minimum n	5214	4976	5084
Avg. Firmware Size (Bytes)	450	450	450
Avg. Compile Time (Sec)	6	5	8
Evaluation Time (Sec)	2475	2610	2836
CPA *p-value*	0.9241	9.376e-7	5.863e-7
DPA *p-value*	0.6427	6.527e-6	7.874e-6
Leakdown Test	Pass	Fail	Fail

F. Evaluation on Mitigated Implementations

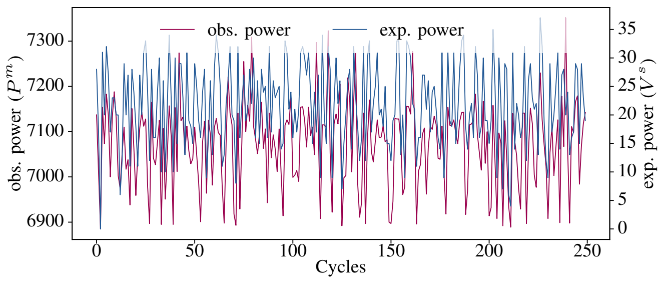
Based on our experiments, we have explored two mitigation strategies: (i) **operand blinding** and (ii) **affine masking**. Operand blinding introduces minimal hardware overhead and has the least impact on the cryptographic instruction set extensions (CISE) workflow. However, it may remain vulnerable to higher-order differential power analysis (DPA), where



(a) Evaluating SCARV-SoC with SHA256



(b) Evaluating SCARV-SoC with AESMIX



(c) Evaluating SCARV-SoC with AESSUB

Fig. 15: Visual (dis)similarity between the observed peak power consumption (■ `obs. power`) of the SoC and the input-based expected power values (■ `exp. power`) of the System evaluation with functional unit prototypes from XCRYPTO cryptographic instruction set extensions. Here the system consists of SCARV-SoC with SCARV-CPU that integrates XCRYPTO functional units.

adversaries analyze combinations of leakage from multiple points in the execution. In contrast, affine masking is a more robust countermeasure for higher-order attacks by applying an affine transformation to the data, which involves both linear and random elements. [45]. However, affine masking comes with a higher hardware overhead, particularly in terms of logic complexity compared to operand blinding. To evaluate the effectiveness of the proposed approach on hardware implementing power side-channel mitigation, we performed two separate experiments with hardware modifications addressing leaking registers in the functional units. In the first experiment, we applied operand blinding by introducing random masks to the operands in the XCRYPTO AESMIX and AESSUB operations, masking the data before performing cryptographic functions and unmasking the results afterward. In the second experiment, we implemented affine masking, where the operand data was masked using a combination of a random value and an affine transformation.

Table V illustrates the summary of the experiments against three SoC configurations of the system with original, operand blinded, and affine mask applied functional units for XCRYPTO AESMIX and AESSUB. During this experiment, we observed no CPA correlation (*p-value* ≈ 1) between the input-based expected power values against the observed power side-channel signature for both operand blinded and affine masked implementations. However, during the analysis with DPA, which accounts for higher order evaluations, operand-blinded implementations failed with *p-value* < 0.05. This illustrates that affine-masked implementation is resistant to potential power side-channel vulnerabilities including higher-order analysis after fabrication.

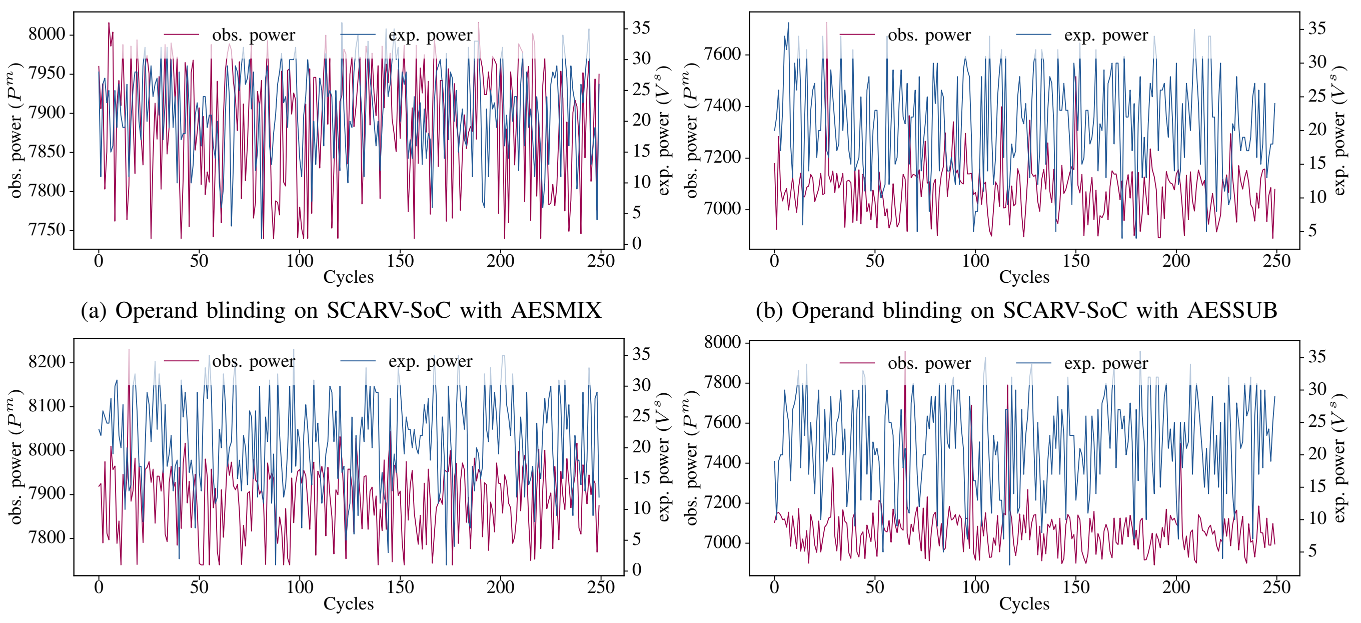
TABLE V: Test vector leakage assessment results on full system prototypes of the original XCRYPTO extensions compared with the modified implementations that incorporate operand blinding power side-channel mitigation.

Functional Unit		**Area %**	**p-value**		**Leakdown Test**
			CPA	**DPA**	
AES MIX	Original	1	7.823e-7	8.342e-7	Fail
	Op. Blind	1.04	0.8720	4.827e-3	Fail
	Affine	1.21	0.9653	0.7521	Pass
AES SUB	Original	1	7.921e-7	8.310e-7	Fail
	Op. Blind	1.05	0.9021	5.924e-3	Fail
	Affine	1.25	0.9672	0.6739	Pass

Figure 16 illustrates the visual dissimilarity between the observed power consumption (■ `obs. power`) of the SoC and the input-based expected power consumption of the system (■ `exp. power`) after performing operand blinding and affine masking. It can be observed that there are no visual similarities between the observed power consumption and the input-based expected power consumption, however, higher-order analysis confirms the potential for information leakage with operand blinding mitigation as illustrated by the results of Table V. Although the mitigation strategies discussed provide protection against higher-order analysis, it is important to use true random number generators (TRNG) during the fabrication. This experiment illustrates that our pre-silicon side-channel evaluation framework is universally applicable regardless of whether the design incorporates mitigation or not. In fact, a design should use our framework after incorporating mitigation to ensure that the mitigation is effective.

VI. Applicability and Limitations

As discussed in Section I, there are two ways of designing fast cryptographic implementations: hardware accelerators and cryptographic instruction set extensions (CISE). This paper focused on the information leakage assessment of cryptographic instruction set extensions (CISE) prototypes. However, our proposed framework can be extended to support the evaluation of hardware accelerators as well as cryptographic coprocessors with minor modifications. For example, when we want to evaluate an AES hardware accelerator which is connected using a memory-mapped input/output (MMIO) interface, the system evaluation round should be modified with necessary firmware modifications. Additionally, the focus of our work is on pre-silicon evaluation methodologies tailored specifically for early-stage design exploration, in line with state-of-the-art pre-silicon side-channel evaluation techniques. At this



(a) Operand blinding on SCARV-SoC with AESMIX (b) Operand blinding on SCARV-SoC with AESSUB

(c) Affine Masking on SCARV-SoC with AESMIX (d) Affine Masking on SCARV-SoC with AESSUB

Fig. 16: Visual dissimilarity between the bserved peak power consumption (🟥 `obs. power`) of the SoC and the input-based expected power values (🟦 `exp. power`) after performing power side-channel mitigation on functional modules.

stage, our objective is to identify cryptographic weaknesses and potential leakage vulnerabilities at the architectural and logic levels of cryptographic instruction set extension (CISE) prototypes, before considering the physical effects associated with post-routing concerns.

Similar to existing TVLA methods, our approach assumes the knowledge of potentially vulnerable cryptographic functions (e.g., cryptographic collisions) as well as module (unit) level boundaries. Although there are many industrial CISE implementations with different instruction set architectures, we do not have access to the corresponding hardware to apply our proposed framework. As a result, we applied only on open-source CISE prototype implementations.

Cryptographic instruction set extensions (CISE) ecosystem consists of many components and interactions between them, including custom instructions, compilers, firmware templates, hardware modules, and validation framework with the continuous integration and continuous deployment (CICD) pipeline. Currently, the validation framework uses an effective combination of simulation-based validation and formal methods. Going forward, our proposed information leakage assessment framework will be included into the CICD pipeline.

VII. Conclusion

Cryptographic instruction set extensions (CISE) is a promising avenue to design fast and flexible security implementation. Unfortunately, there are many demonstrated attacks on CISE implementations and it is hard to mitigate them without introducing significant performance overhead. Clearly, there is a need to develop an efficient solution for verifying the existence of side-channel vulnerabilities in CISE prototypes. In this paper, we proposed a test vector leakage assessment framework that can be used to evaluate information leakage in hardware implementations of CISE prototypes. Specifically, this paper made three important contributions. First, we evaluate each functional unit for potential power side-channel leakage of internal secrets. Next, if the functional units are determined to be leaky, we also evaluate the system model for potential power side-channel leakage. Finally, we have demonstrated the applicability and effectiveness of our proposed framework using two CISE prototypes, RISCV-CRYPTO and XCRYPTO, covering eight functional units of fu_ssha256, fu_ssha512, xc_sha256, xc_sha512, fu_saes32, fu_saes64, xc_aesmix and xc_aesmix. Experimental results revealed that, except for the full system evaluation of xc_sha256, all other functional modules along with their systems evaluations failed the leak-down test, signifying that there is a considerable amount of information leakage during the computations of cryptographic workloads. These results also highlight the need for pre-silicon test vector leakage assessment during the development life-cycle of cryptographic instruction set extensions.

Acknowledgments

This work was partially supported by the National Science Foundation (NSF) grant CCF-1908131.

References

[1] A. Arapov, D. Misharov, H. Landau, J. Muir *et al.*, "Openssl cryptography and ssl/tls toolkit," https://www.openssl.org/, 2023, accessed: 2023-8-10.

[2] "wolfcrypt embedded crypto engine," https://www.wolfssl.com/products/wolfcrypt-2/, 2023, accessed: 2023-8-10.

[3] W. Koch and M. Schulte, "The libgcrypt reference manual," *Free Software Foundation Inc*, pp. 1–47, 2005.

[4] W. Dai, "Crypto++ library is a free c++ class library of cryptographic schemes," https://www.cryptopp.com/, 2023, accessed: 2023-8-10.

[5] "Titan in depth: Security in plaintext," https://cloud.google.com/blog/products/identity-security/titan-in-depth-security-in-plaintext, 2023, accessed: 2023-8-10.

[6] "Ibm pcie cryptographic coprocessor," https://www.ibm.com/products/pcie-cryptographic-coprocessor, 2023, accessed: 2023-8-10.

[7] "Stmicroelectronics: Trusted platform module," https://www.st.com/en/applications/embedded-security/trusted-platform-module.html, 2023, accessed: 2023-8-10.

[8] "Optiga™ tpm - trusted platform module," https://www.infineon.com/cms/en/product/security-smart-card-solutions/optiga-embedded-security-solutions/optiga-tpm/, 2023, accessed: 2023-8-10.

[9] S. Gueron, "Intel advanced encryption standard (intel aes) instructions set–rev 3.01," *Intel, Aug*, 2012.

[10] S. Gulley, V. Gopal, K. Yap, W. Feghali, J. Guilford, and G. Wolrich, "Intel sha extensions," https://www.intel.com/content/dam/develop/external/us/en/documents/intel-sha-extensions-white-paper.pdf, 2023, accessed: 2023-8-10.

[11] "Arm cryptography extension," https://developer.arm.com/documentation/ddi0501/f/introduction/about-the-cortex-a53-processor-cryptography-extension, 2023, accessed: 2023-8-10.

[12] A. Jayasena, E. Andrews, and P. Mishra, "TVLA*: Test Vector Leakage Assessment on Hardware Implementations of Asymmetric Cryptography Algorithms," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2023.

[13] M. He, J. Park, A. Nahiyan, A. Vassilev, Y. Jin, and M. Tehranipoor, "RTL-PSC: Automated power side-channel leakage assessment at register-transfer level," in *IEEE VLSI Test Symposium (VTS)*, 2019, pp. 1–6.

[14] N. Pundir, J. Park, F. Farahmandi, and M. Tehranipoor, "Power side-channel leakage assessment framework at register-transfer level," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2022.

[15] RISC-V Cryptography Working Group, "RISC-V Cryptography Library," https://github.com/riscv/riscv-crypto, 2023.

[16] SCARV, "XCrypto: Cryptographic Library," https://github.com/scarv/xcrypto, 2023.

[17] J. K Rott, "Intel® advanced encryption standard instructions (aes-ni)," https://www.intel.com/content/www/us/en/developer/articles/technical/advanced-encryption-standard-instructions-aes-ni.html, 2023, accessed: 2023-8-10.

[18] G. Lupescu, L. Gheorghe, and N. Tapus, "Commodity hardware performance in aes processing," in *2014 IEEE 13th International Symposium on Parallel and Distributed Computing*. IEEE, 2014, pp. 82–86.

[19] ARM. (June 2023) SIMD and Floating-Point Extensions: SHA1C - SHA1 hash update (choose). https://developer.arm.com/documentation/ddi0597/2023-06/SIMD-FP-Instructions/SHA1C--SHA1-hash-update--choose--?lang=en.

[20] R. Newell and M. Xing, "Scalar crypto standardization status summary," https://wiki.riscv.org/display/HOME/Scalar+Crypto+Standardization+Status+Summary, 2023, accessed: 2023-8-10.

[21] J. R. Kiniry, D. M. Zimmerman, R. Dockins, and R. Nikhil, "A formally verified cryptographic extension to a risc-v processor," *Computer Architecture Research with RISC-V–CARRV 2018*, 2018.

[22] B. Marshall, G. R. Newell, D. Page, M.-J. O. Saarinen, and C. Wolf, "The design of scalar aes instruction set extensions for risc-v," *Cryptology ePrint Archive*, 2020.

[23] B. Marshall, D. Page, and T. Pham, "Implementing the draft risc-v scalar cryptography extensions," in *Hardware and Architectural Support for Security and Privacy*, 2020, pp. 1–8.

[24] H. Cheng and D. Page, "eliminate: a leakage-focused ise for masked implementation," *Cryptology ePrint Archive*, 2023.

[25] Q. Bao, Z. Wang, X. Li, J. R. Larus, and D. Wu, "Abacus: Precise side-channel analysis," in *IEEE/ACM International Conference on Software Engineering (ICSE)*, 2021, pp. 797–809.

[26] M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss, "Platypus: Software-based power side-channel attacks on x86," in *IEEE Symposium on Security and Privacy*, 2021, pp. 355–371.

[27] G. Haas and A. Aysu, "Apple vs. ema: electromagnetic side channel attacks on apple corecrypto," in *ACM/IEEE Design Automation Conference*, 2022, pp. 247–252.

[28] W. R. Cordwell, "Side channel considerations for sha-512," 7 2020. [Online]. Available: https://www.osti.gov/biblio/1647526

[29] Y. Belenky, I. Dushar, V. Teper, V. Bugaenko, O. Karavaev, L. Azriel, and Y. Kreimer, "Carry-based differential power analysis (cdpa) and its application to attacking hmac-sha-2," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 1–29, 2023.

[30] B. J. Gilbert Goodwill, J. Jaffe, P. Rohatgi *et al.*, "A testing methodology for side-channel resistance validation," in *NIST non-invasive attack testing workshop*, vol. 7, 2011, pp. 115–136.

[31] A. Jayasena and P. Mishra, "Directed test generation for hardware validation: A survey," *ACM Computing Surveys*, 2023.

[32] T. Zhang, J. Park, M. Tehranipoor, and F. Farahmandi, "PSC-TG: RTL power side-channel leakage assessment with test pattern generation," in *ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 709–714.

[33] A. Jayasena, R. Bachmann, and P. Mishra, "Evilcs: An evaluation of information leakage through context switching on security enclaves," in *Design Automation & Test in Europe*, 2024, pp. 1–6.

[34] S. Collin and F.-X. Standaert, "Side channel attacks against the solo key-hmac-sha256 scheme," Ph.D. dissertation, Ph. D. thesis, UCL-Ecole polytechnique de Louvain, 2020.

[35] D. E. Knuth, "Computer programming as an art," in *ACM Turing award lectures*, 2007, p. 1974.

[36] D. Shanmugam and P. Schaumont, "Improving side-channel leakage assessment using pre-silicon leakage models," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, 2023, pp. 105–124.

[37] R. Sadhukhan, P. Mathew, D. B. Roy, and D. Mukhopadhyay, "Count your toggles: A new leakage model for pre-silicon power analysis of crypto designs," *Journal of Electronic Testing*, vol. 35, pp. 605–619, 2019.

[38] A. Srivastava, S. Das, N. Choudhury, R. Psiakis, P. H. Silva, D. Pal, and K. Basu, "Scar: Power side-channel analysis at rtl level," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2024.

[39] "Synopsys PrimePower," https://www.synopsys.com/implementation-and-signoff/signoff/primepower, accessed: 2024-10-19.

[40] "Cadence Voltus," https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/silicon-signoff/voltus-ic-power-integrity-solution, accessed: 2024-10-19.

[41] "Siemens PowerPro," https://eda.sw.siemens.com/en-US/ic/powerpro/rtl-power-estimation/, accessed: 2024-10-19.

[42] P. Kocher, "Differential power analysis," in *Proc. Advances in Cryptology (CRYPTO'99)*, 1999.

[43] F.-X. Standaert, E. Peeters, and J.-J. Quisquater, "On the masking countermeasure and higher-order power analysis attacks," in *International Conference on Information Technology: Coding and Computing (ITCC'05)-Volume II*, vol. 1. IEEE, 2005, pp. 562–567.

[44] W. Snyder, "Verilator: Open simulation-growing up," *DVClub Bristol*, 2013.

[45] G. Fumaroli, A. Martinelli, E. Prouff, and M. Rivain, "Affine masking against higher-order side channel analysis," in *International Workshop on Selected Areas in Cryptography*. Springer, 2010, pp. 262–280.

Aruna Jayasena is a Ph.D student in the Department of Computer & Information Science & Engineering at the University of Florida. He received his B.S. in the Department of Computer Science and Engineering at the University of Moratuwa, Sri Lanka, in 2019. His research focuses on systems security, hardware-firmware co-validation, applied cryptography, trusted execution, side-channel analysis, and test generation.

Richard Bachmann received a B.S. in computer science from the University of Florida in 2023. He is currently a software developer at the MIT Lincoln Laboratory. His research interests include embedded systems, formal methods, programming languages, and software design complexity.

Prabhat Mishra is a Professor in the Department of Computer and Information Science and Engineering at the University of Florida. He received his Ph.D. in Computer Science from the University of California at Irvine. His research interests include embedded systems, hardware security, formal verification, system-on-chip validation, machine learning, and quantum computing. He is an IEEE Fellow, an AAAS Fellow, and an ACM Distinguished Scientist.