An Open-Source Framework for Efficient Side-Channel Analysis on Cryptographic Implementations

Takuya Kojima *Member, IEEE*, Masaki Morita, Hideki Takase, and Hiroshi Nakamura *Senior Member, IEEE*

***Abstract*—Side-channel attacks (SCAs) are increasingly recognized as a significant threat to hardware roots of trust. As a result, cryptographic module designers must ensure that their modules are resilient to such attacks before deployment. However, efficient evaluation of side-channel vulnerabilities in cryptographic implementations remains challenging. This article introduces an open-source framework integrating field-programmable gate array (FPGA) designs, power measurement tools, and high-performance side-channel analysis libraries to streamline the evaluation process. The framework provides design templates for two widely used FPGA boards in the side-channel analysis community, enabling Shell-Role architecture, a modern FPGA design pattern. This shell abstraction allows designers to focus on developing cryptographic modules while utilizing standardized software tools for hardware control and power trace acquisition. Additionally, the framework includes acceleration plugins for ChipWhisperer, the leading open-source side-channel analysis platform, to enhance the performance of correlation power analysis (CPA) attacks. These plugins exploit modern many-core processors and graphics processing units (GPUs) to speed up analysis significantly. To showcase the capabilities of the proposed framework, we conducted multiple case studies and highlighted significant findings that advance side-channel research. Furthermore, we compare our CPA plugins with existing tools and show that our plugins achieve up to 8.60× speedup over the state-of-the-art CPA tools.**

***Index Terms*—Side-channel attack, Hardware security, GPU acceleration**

I. Introduction

RECENT computing platforms rely on cryptographic modules to ensure the confidentiality and integrity of sensitive information. For example, modern operating systems (e.g., Windows 11) require a dedicated hardware module, a Trusted Platform Module (TPM) 2.0 chip, for cryptographic primitives. This decreases the risk of malicious software [1], while a cryptographic module can still be vulnerable physically. Side-channel attack (SCA) is a class of attacks that exploit the physical implementation of a cryptographic module to extract secret information, such as secret keys. An adversary exploits the physical characteristics of the cryptographic module, such as power consumption, to recover the secret information. In particular, power analysis attacks [2] have emerged as a critical threat, and thus, their countermeasures have been widely studied [3].

T. Kojima is with Department of Computer Science, Institute of Systems and Information Engineering, University of Tsukuba, Ibaraki (e-mail: tkojima@lila.cs.tsukuba.ac.jp). M.Morita, H.Takase, and H.Nakamura are with Graduate School of Information Science and Technology, The University of Tokyo, Tokyo. This work was partially supported by JST PRESTO, Japan, Grant Number JPMJPR22P5 and JSPS KAKENHI Grant Numbers 25K21171 and 23K28051.

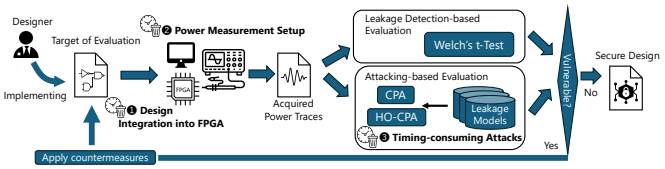


Fig. 1: General vulnerability assessment flow for secure cryptographic designs against SCAs and three obstacles to efficient evaluation.

SCAs exploit vulnerabilities in a hardware or software implementation rather than flaws in the cryptographic algorithm itself. Therefore, the pre-silicon vulnerability assessment of a cryptographic module against SCAs is essential to ensure its security. Two major standards, common criteria (CC) [4] and ISO 17825:2016 [5], provide comprehensive guidelines for the security evaluation of cryptographic modules. Each guideline is based on different strategies for assessing the resistance to SCAs [6], [7]. CC employs a rating system that evaluates the cost of successful attacks by considering various factors, such as elapsed time, the number of available device samples, and the need for special equipment. Therefore, the evaluator must repeat the attack process for all possible scenarios, which takes time and effort.

ISO 17825:2016 recommends using the test vector leakage assessment (TVLA) [8]. The TVLA employs Welch's t-test, a statistical hypothesis test, to detect potential information leakage in cryptographic modules. Although it is efficient and independent of the implementation details, unlike the CC standard, it cannot guarantee complete immunity against possible attacks. That is why the TVLA is sometimes treated with the first phase of the evaluation process [9]. The evaluator should conduct the vulnerability assessment based on both standards in a complementary manner, as in [10].

Fig. 1 illustrates the general flow of the vulnerability assessment. Field-programmable gate arrays (FPGAs) are the most popular platforms for evaluating the SCA resistance of cryptographic modules [11]. Although simulation-based evaluations at the register-transfer level (RTL) or gate level are often used, they suffer from the prohibitively long simulation time to acquire sufficient traces, generally in the order of millions. In contrast, FPGA-based evaluations overcome these

Author Accepted Manuscript
Accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.
The final published version is available at: https://doi.org/10.1109/TCAD.2026.3654878.

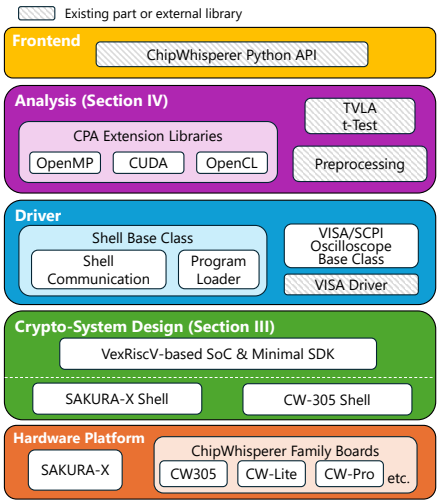


Fig. 2: Proposed framework based on ChipWhisperer

limitations by offering a practical and reliable method to assess leakage, taking into account the physical characteristics of the hardware implementation.

Ideally, designers should concentrate on developing the cryptographic module and implementing robust countermeasures against SCAs. Nevertheless, the evaluation process imposes a significant burden on the designers due to the following supplementary tasks.

1) **Design Integration:** The designer must integrate the cryptographic module into the FPGA from scratch, including communication interfaces with an external host computer.
2) **Power Measurement Setup:** The designer must set up the power measurement equipment, such as an oscilloscope, to work with the FPGA and develop the control software to acquire power traces.
3) **Time-Consuming Attacks:** The designer has to conduct time-consuming attacks, such as correlation power analysis (CPA) [12], high-order CPA (HO-CPA) [13], and deep-learning-based attacks [14], considering any possible leakage data.

Our Approach and Contributions: To alleviate these burdensome tasks for designers and facilitate swift side-channel vulnerability evaluation, we propose an open-source framework [1] integrating FPGA design, power measurement software tools, and fast side-channel analysis libraries, as shown in Fig. 2. The proposed framework provides the following features.

Efficient Design Integration Using FPGA Shells: The proposed framework leverages FPGA shells tailored for two widely used boards in the side-channel analysis community: SAKURA-X [15] and CW305 [16]. While the shell part, which is noncryptographic, might raise concerns about its impact on the accuracy of evaluating the cryptographic module, our experimental results demonstrate negligible influence. To further enhance usability, the framework includes RISC-V system-on-chip (SoC) designs based on the shell templates, enabling the evaluation of software-based countermeasures.

[1]The proposed framework is available at https://github.com/hal-lab-u-tokyo/chipwhisperer-enhanced-plugins

Minimized Effort for Power Measurement Setup: Our framework provides a base implementation of the driver software to control the FPGA designs, including support for data communication. It allows the designer to focus on the specific part of the control software. We have designed the driver software to work with ChipWhisperer [16] APIs, the most successful open-source side-channel analysis framework. Thus, other ChipWhisperer family boards, such as ChipWhisperer-Lite, are available with our framework. We also provide a base implementation for oscilloscopes controlled with the VISA/SCPI protocol to ease the hardware restriction of power measurement. The evaluator can even extend it to support other oscilloscopes by defining vendor-specific commands.

Many-Core/GPU-Accelerated CPA Plugins:

We provide plugins compatible with the ChipWhisperer to accelerate the CPA evaluation and exploit modern many-core processors and graphics processing units (GPUs). To support the four major GPU vendors (NVIDIA, AMD, Intel, and Apple), we provide an OpenCL-based implementation as well as a CUDA-based implementation only for NVIDIA GPUs. Our GPU-accelerated CPA library can achieve about 890× speedup with a mid-range NVIDIA GPU compared to the original Python implementation. Besides, this work shows that our plugins are faster or comparable to the state-of-the-art CPA tools.

Furthermore, we conducted three experiments using the proposed framework. For the first experiment, we thoroughly evaluated the impact of the instruments for power measurement, such as the operational frequency of cryptographic modules and the sampling rate of oscilloscopes. After the evaluation, we suggested an appropriate setting for the power measurement equipment to ensure an accurate assessment of the cryptographic module. For the second experiment, we investigated the effect of compiler optimizations on software-based countermeasures using the RISC-V SoC design and explored secure coding practices to mitigate the risk of information leakage. For the last experiment, we demonstrated the potential risks associated with HLS-implemented hardware countermeasures, where automatic optimizations cause unintentional information leakage.

The remainder of this article is organized as follows. Section II explains a brief overview of SCAs and related work. Section III and Section IV describe the proposed FPGA shell designs and the accelerated CPA library, respectively. Then, we present the experimental results in Section V. Finally, we conclude the article in Section VI.

II. Background and Motivation

A. Threat Models

SCAs can be categorized based on the attacker's capabilities and the type of side-channel information exploited. If the attacker has a fully controllable device identical to the target device, they can perform profiling attacks. Template attacks [17] are a well-known example of profiling attacks, where the attacker builds statistical models of the device's leakage behavior during a profiling phase and uses these models to extract secret information during the attack phase.

Beyond such classical models, powerful attacks leveraging machine learning techniques have emerged as a significant threat [14]. However, considering the practical scenarios, nonprofiling attacks are more serious threats because they do not require such extensive control over the target device. CPA [12] is a widely recognized nonprofiling attack method. A few works based on machine learning techniques also target nonprofiling scenarios [18]. In this work, we focus on the nonprofiling attacks, particularly CPA, due to its practicality and relevance in real-world scenarios. However, part of our framework, such as the FPGA shell and power measurement setup, can also be utilized for profiling attacks, including template attacks and machine learning-based attacks.

B. Correlation Power Analysis

In a CPA attack, the attacker first measures power consumption waveforms during cryptographic operations and records them as traces. After acquiring thousands to millions of traces, the attacker analyzes them to recover the secret information. The attacker identifies the most likely key candidate based on the correlation between the power consumption and the hypothetical intermediate value processed inside the cryptographic module. Although power consumption is modeled in various ways, such as at the analog and logic level, the CPA usually employs more simplified models, which assume that the power consumption is a linear function of the number of "1" bits in the intermediate value of the cryptographic hardware [2]. CMOS transistors consume dynamic power by charging and discharging the load capacitance when transients occur. In this respect, if bits related to sensitive information are changed from v_0 to v_1, the Hamming-distance model defines the power consumption as $HD(v_0, v_1)$, where HD is the Hamming-distance function. In contrast, the Hamming weight model is more straightforward and only considers a single state of the intermediate value as $HW(v_0)$. Instead of power consumption measurements, electromagnetic (EM) emanations can also be exploited in a similar manner.

Advanced encryption standard (AES) is a widely used symmetric-key encryption algorithm, and its vulnerabilities to SCA have been extensively studied [3]. Several candidates for the intermediate value exist in the case of AES. The most common one is the output of the S-box, the nonlinear substitution function. The AES algorithm with a 128-bit key comprises ten rounds, and *KeySchedule* generates 11 round keys from the original key. The generated round keys are XORed at the beginning of each round. Each byte of the XORed value is individually passed to the S-box. Although every round of S-box outputs are candidates of the intermediate value, the first round is the most common choice. Given a candidate subkey $\hat{k}$ and corresponding plaintext byte p, hypothetical leakage $h_{\hat{k}}$ is represented as follows:

$$h_{\hat{k}} = HW(SBox(p \oplus \hat{k})) \tag{1}$$

CPA calculates the Pearson correlation coefficient between the power consumption and the hypothetical leakage. The correlation coefficient ρ between sample vectors X and Y is defined as follows:

$$\rho(X,Y) = \frac{Cov(X,Y)}{\sqrt{Var(X)Var(Y)}} \tag{2}$$

where Cov denotes the covariance, and Var denotes the variance.

The CPA attack considers X and Y as the power consumption and the hypothetical leakage, respectively. The waveform for each trace is composed of L samples, and let $w_{i,t}$ be the ith sample of the tth trace. In addition, the leakage model depends on the hypothetical subkey byte $\hat{k}$, which can be any value from 0 to 255 for AES-128. Therefore, it is necessary to calculate the correlation coefficient for all candidates and all sample positions. The subkey byte corresponding to the highest correlation coefficient is identified as the most likely candidate for the target subkey.

Given the hypothetical subkey byte $\hat{k}$ and the sample position t, the correlation coefficient $\rho_{\hat{k},t}$ is calculated as follows:

$$\rho_{\hat{k},t} = \frac{\sum_{i=1}^{N}[(h_{i,\hat{k}} - \bar{h}_{\hat{k}}) \cdot (w_{i,t} - \bar{w}_t)]}{\sqrt{\sum_{i=1}^{N}(h_{i,\hat{k}} - \bar{h}_{\hat{k}})^2}\sqrt{\sum_{i=1}^{N}(w_{i,t} - \bar{w}_t)^2}} \tag{3}$$

$$= \frac{N\sum_{i=1}^{N} h_{i,\hat{k}} w_{i,t} - \sum_{i=1}^{N} h_{i,\hat{k}} \sum_{i=1}^{N} w_{i,t}}{\sqrt{(\sum_{i=1}^{N} h_{i,\hat{k}})^2 - N\sum_{i=1}^{N} h_{i,\hat{k}}^2}\sqrt{(\sum_{i=1}^{N} w_{i,t})^2 - N\sum_{i=1}^{N} w_{i,t}^2}} \tag{4}$$

where N is the total number of traces. The first equation is the naive form of the correlation coefficient calculation. Since the average calculation is required, all the traces and hypothetical leakages must be summed up in advance. In contrast, the second equation is optimized for online calculation [19]. It also allows calculating the correlation coefficient progressively, updating each summation with partial traces. This characteristic is beneficial for GPU acceleration since the global memory of the GPU has limited capacity compared to the main memory of a host processor.

C. Related Work

Most prior works on side-channel analysis tools focus on either the analysis software or the hardware platform. Table I summarizes the existing works. The first four works focus on the analysis software, while the latter works include hardware platforms, such as oscilloscopes and evaluation boards, in addition to the analysis software.

1) Efficient CPA Tools: As the number of traces N and the length of the trace L increase, the CPA attack becomes more time-consuming. Thus, the number of traces necessary to recover the key is one of the security metrics of the cryptographic module [27]. To boost the assessment cycle of the resistance to SCA, it is crucial to speed up the calculation of the correlation coefficients. CPA is a univariate analysis where the correlation coefficient is calculated independently for each sample position. On the contrary, HO-CPA [13] is a multivariate analysis that calculates the correlation coefficient combining multiple sample positions. Although dth-order CPA can theoretically break $(d-1)$th-order masking countermeasures, its practical feasibility decreases significantly for $d > 2$ due to the increasing noise effect [28]. Therefore, the second-order CPA (SO-CPA) ($d = 2$) is widely investigated. Even for

TABLE I: Comparison of existing frameworks/tools/methods and the proposed framework

Framework/Paper	OSS	Analysis/Attack Software			Hardware	
		Method	Model[1]	Implementation[2]	Target	Trace
RamDPA [20]	✓	CPA	✗	C++ (MT)		✗
DareDevil [21]	✓	CPA, HO-CPA	–	C++ (MT)		✗
T.Swamy *et al.* [22]	✗	CPA	✗	CUDA		✗
H.Gamaarachchi *et al.* [23]	✗	CPA	✗	OpenCL, CUDA		✗
SICAK [24]	✓	CPA, HO-CPA t-test	✓	OpenMP, OpenCL	✗	PicoScope 6000 VISA oscilloscopes
FOBOS3 [25]	✓	CPA, TVLA	✓	Python	FBD-A7, etc	FOBOS Shield
SASEBO/SAKURA[26]	✓	CPA	✗	C# (MT)	SAKURA-series	VISA oscilloscopes
ChipWhisperer[16]	✓	CPA, TVLA + Template	✓	Python, C++	CW-series	CW-series
This work	✓	CPA, SO-CPA CW features	✓	Python with OpenMP, CUDA & OpenCL	FPGA Shells[3] + CW-series	VISA oscilloscopes + CW-series

1: ✓: any model available, –: partially flexible, and ✗: fixed model, 2: MT: multi-threading, 3: Currently, the FPGA shells are available for SAKURA-X and CW305 boards.

SO-CPA, correlation coefficients need to be calculated up to $L(L-1)/2$ times more than the first-order CPA.

ChipWhisperer [16] is an open-source and widely used SCA framework primarily written in Python. It offers several practical leakage models for CPA attacks and the flexibility to use a custom leakage model. However, it suffers from the performance issue of the CPA attack because of the Python-based implementation. Although it has a C++ implementation of a time-consuming part of the CPA attack, it does not consider parallelization and reuse of the partial results.

Several works have proposed acceleration of the CPA attack. RamDPA [20] is a multithreaded implementation of the CPA attack. It presents a cache-friendly data layout to reduce the performance overhead attributed to the cache coherence problem. Their tool is published as an open-source software, yet users need to convert trace data to its format to use with the tool. In addition, the leakage model has no flexibility. DareDevil [21] is a multithreaded implementation of the CPA attack to accelerate the CPA attack with a multicore processor and is available as open-source software. It supports HO-CPA as well as the traditional first-order CPA. However, it also employs a custom data format for traces similar to RamDPA. Even though it allows users to define their leakage model by creating look-up tables of leakage values, Hamming-distance models are unavailable.

A few works tackle the issue of the time-consuming CPA attack with GPU acceleration [22], [23]. Swamy *et al.* [22] present both CUDA and OpenCL implementations of the CPA attack. However, their analysis algorithm slightly differs from the usual CPA attack explained in Section II-B. Gamaarachchi *et al.* [23] also provide GPU acceleration but only target NVIDIA GPUs with CUDA. Although the GPU part of their implementation is similar to ours, we propose a hybrid acceleration approach with OpenMP and GPUs. The communication cost between a host processor is not negligible in most cases. That is why the processor should handle some parts of the process that are unsuitable for GPU acceleration.

Among the GPU-accelerated CPA tools, SICAK [24] stands out as the only one available as open-source software. In addition, it includes several plugins, such as oscilloscope control. However, compared to our work, it has limited support for hardware platforms. Note that Socha *et al.* [29], who are contributors to SICAK, also presented a GPU-accelerated HO-CPA implementation, whereas the published software includes only many-core acceleration for HO-CPA. Furthermore, the precision of the correlation coefficient calculation is often overlooked. SICAK, for instance, assumes 16-bit integer values for the power traces and employs double-precision floating-point arithmetic for calculating the correlation coefficients. However, this approach leads to inaccuracies in the correlation coefficient calculation and limits the available GPUs. For example, some recent GPUs (e.g., AppleSilicon GPU [30]) do not support double-precision floating-point arithmetic.

In summary, the aforementioned acceleration-focused works have limitations such as inflexibility in the leakage model, non-standard trace data formats, limited hardware platform support, and less consideration for the precision of the correlation coefficient calculation. In contrast, our proposed framework extends ChipWhisperer, leveraging its comprehensive features. Our framework accelerates the CPA attack using OpenMP, CUDA, and OpenCL while maintaining compatibility with ChipWhisperer's interface. We focus on the SO-CPA attack rather than arbitrary-order analysis because of the aforementioned practical feasibility and offer its acceleration with GPUs as well as multicore processors. Our implementation pays attention to the precision of the correlation coefficient calculation and employs a software emulation technique that provides extended precision for some platforms, as explained in Section IV. Similar to ChipWhisperer, available leakage models are flexible, while the hypothetical leakage values must be determined with plaintext, ciphertext, and key guesses.

2) Hardware Platforms for Side-Channel Analysis: Three different projects, FOBOS3 [25], SASEBO/SAKURA [26], and ChipWhisperer [16], offer evaluation boards for FPGA-based side-channel analysis. The FOBOS3 FBD-A7 board is a fully open-source design, yet it is not commercially available. The SASEBO project and the later project, SAKURA, provide several FPGA boards, such as SASEBO-GII, SAKURA-G, and SAKURA-X. Some are commercially available and widely

used in academia. FOBOS3 also provides FOBOS Shield as a power measurement device, whereas the SASEBO/SAKURA family boards rely on third-party oscilloscopes

ChipWhisperer [16] offers not just the analysis tools but also other inevitable primitives for SCA research, such as attack target hardware, waveform acquisition, and preprocessing of the acquired waveforms. This project produces several commercially available products. For example, the ChipWhisperer-Nano board integrates an analog–digital converter for waveform acquisition and an ARM Cortex-M0 for evaluating software implementations of cryptographic algorithms. Another product, the ChipWhisperer CW305 board with Xilinx Artix-7 FPGA, is designed for assessing hardware implementations of cryptographic algorithms.

In terms of hardware design on the FPGAs, all the above-mentioned boards have little support for the quick implementation of an evaluation target. Each of them provides some reference designs, such as AES encryption, yet replacing the target cryptographic module with a custom one is not straightforward. This is because its communication interface is not standardized. Thus, designers may need to spend a fair amount of effort to implement their target cryptographic module for each board.

A design pattern known as *shell-role architecture* is gaining popularity in recent FPGA designs (e.g., AWS F1 instances) [31]. In this design style, the FPGA design is divided into *shell* and *role* regions. The shell contains predefined and commonly used modules, such as DDR memory interfaces and clock management, whereas user logic (i.e., application-specific logic) is placed in the role region. Both regions are usually connected through standardized interfaces. Thereby, it abstracts the board-specific features and increases the reusability of the user logic design.

Based on the concept of the shell-role architecture, our framework provides FPGA shells for the SAKURA-X and CW305 boards, facilitating a modern FPGA design approach. Thus, the identical IP cores can be evaluated for both boards. Our FPGA shell has an AXI4 interface for the user logic and encapsulates the board-specific communication protocols, as described in Section III. It enables one to evaluate HLS designs of cryptographic algorithms on the FPGA boards with minimal effort. The HLS implementation is a promising approach to obtaining highly optimized hardware designs, especially for complicated cryptographic algorithms. Hence, our FPGA shells are expected to play a key role in upcoming side-channel analysis research. Additionally, the ready/valid handshaking of the AXI4 interface supports designs with cross-clock domain crossing and variable latency data responses, which is beneficial for evaluating countermeasures like random dynamic frequency scaling [32], [33].

AXI4 protocol can be easily translated to other standard protocols. For example, some of the bridge IPs, such as AXI4-AHB, are provided by AMD Xilinx. It enables us to evaluate Adam's Bride targeted in [34], which is an open-source IP core for post-quantum cryptographic algorithms with an AHB interface. Although our framework does not provide acceleration libraries for other SCAs than CPA, ChipWhisperer initially offers several analysis methods, such as template attacks and TVLA. Thanks to the compatibility with ChipWhisperer, designs based on our FPGA shells can also be evaluated using these methods. In addition, our other plugin is available for machine-learning-based side-channel analysis libraries [35].

III. Proposed FPGA Shell

A. FPGA Shell for SAKURA-X

The SAKURA-X board is composed of two FPGAs: (1) Xilinx Spartan-6 and (2) Kintex-7. Kintex-7 is used for cryptographic implementations, while Spartan-6 is used for the control logic and the communication interface. For power-analysis attack evaluation, the SAKURA-X board includes a dedicated power measurement point designed to capture the power consumption exclusively from the Kintex-7 FPGA.

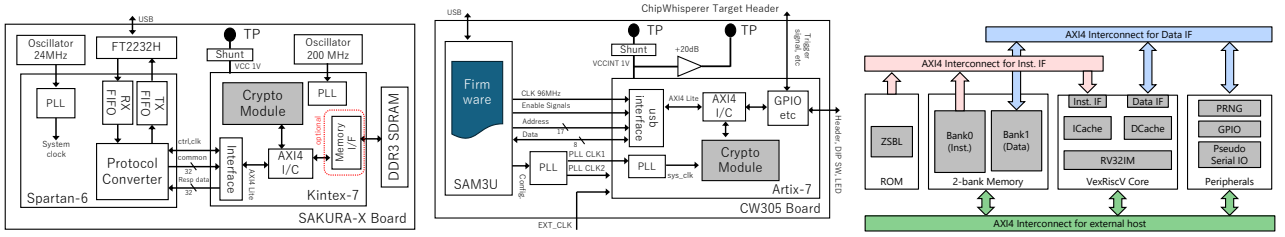
The shell abstraction for SAKURA-X is shown in Fig. 3(a). The shell standardizes the functionality of Spartan-6 to translate data access requests from the host PC into compatible data access requests for Kintex-7. Those requests are converted to the AXI4-Lite protocol on the Kintex-7 side due to the limited interconnectivity between Spartan-6 and Kintex-7. To facilitate ready/valid handshaking, the Spartan-6 FPGA incorporates FIFO buffers and a state machine to manage data flow. Our framework provides high-level APIs for data access based on the shell abstraction. It allows designers to concentrate on developing cryptographic implementations on the Kintex-7 FPGA. Optionally, we extend the shell to utilize the on-board DDR3 memory, enhancing data storage and buffering capabilities.

The shell itself should not consume a significant amount of resources, and in our implementation, it occupies only about 0.14% of the LUTs and 0.29% of the FFs. As mentioned in Section II-C, the evaluated core can operate at its own clock frequency independently of the shell part. Thus, the shell part minimizes its impact on the timing and available resources of the target cryptographic module.

B. FPGA Shell for CW305

Fig. 3(b) shows the shell abstraction for the CW305 board. Similar to SAKURA-X, the CW305 board is specialized to support power measurement and side-channel analysis, whereas the CW305 board is equipped with a single FPGA (Xilinx Artix-7). Instead, a microcontroller, Atmel SAM3U, is responsible for the control logic and communication interface. Although the firmware of the microcontroller is open-sourced, we have developed an FPGA shell without modifying the firmware. However, its firmware assumes that the FPGA responds to read requests from the microcontroller in static clock cycles. This assumption is not suitable for our shell abstraction design with a ready/valid handshaking protocol.

To address this issue, we introduce a packet-based communication protocol over the physical communication layer. Even though the communication interface of the microcontroller has a 17-bit address and 8-bit data buses, we bundle them into a single 25-bit flit to form a packet. The flit is composed of a 16-bit payload, 4-bit data length, 4-bit flit type, and 1-bit parity bit. For example, a read request packet consists of two flits. The 32-bit address is divided into upper and lower



(a) Shell for SAKURA-X board (b) Shell for CW305 board (c) RISC-V SoC based on the shells

Fig. 3: Hardware templates in the proposed framework

16-bit parts, and the upper part is sent first as payload. The lower part is sent as the second payload. The first flit also contains the number of bytes to be read as the data length field. When the shell part on the FPGA side receives the read request packet, it translates the request into a read request for the AXI4-Lite protocol. Simultaneously, the shell part responds to the microcontroller with an ACK packet (not a data response) to indicate that the request has been received. Then, the microcontroller repeatedly checks the status of the FPGA with another type of packet in a polling manner. In this way, the shell and its high-level APIs realize the ready/valid handshaking protocol without modifying the firmware. The shell part consumes around 1% of the LUTs and FFs for CW305 as well.

C. RISC-V SoC for Software Implementation Evaluation

Our framework also provides a RISC-V SoC design, as illustrated in Fig. 3(c). The RV32IM configuration of VexRiscv [36] is used as a RISC-V core, and necessary peripherals are added to the SoC design. Each module is connected to the AXI4 buses dedicated to the cores. In addition, some modules are connected to an AXI4-Lite bus for external access, which is connected to the shell interface, to load the software binary and transfer data such as key, plaintext, and ciphertext. A minimal software development kit (SDK) provides an application development environment for the SoC design and software libraries for the peripherals. A software driver based on the high-level APIs of the shell can boot the SoC and control the peripherals.

IV. CPA Acceleration Library

This section begins by identifying the critical portion of the correlation calculation process in terms of computational cost and opportunities for acceleration. Then, we describe several techniques for accelerating the correlation calculation process with many-core CPUs and GPUs.

A. Profiling and Acceleration Target

To determine the exciting part to be accelerated, we profiled the correlation calculation process with 4000 traces of 1000-sample waveforms using our single-threaded C++ implementation. As explained in Section II-B, it is necessary to calculate the correlation coefficients for each sample position t, each key candidate $\hat{k}$, and additionally for each subkey byte position b. Our preliminary profiling results revealed that the most time-consuming part is the summation of $h_{i,\hat{k}}w_{i,t}$ because its value depends on all $(t, \hat{k}, b)$ and cannot be reused. It forms four nested loops and accounts for more than 99% of the total time and needs to be accelerated.

Regarding the SO-CPA, we employ the same formulation in [21]. As mentioned in Section II-C, the conventional SO-CPA needs to calculate the correlation coefficients for any combination of sample position t. To decrease the number of correlation calculations, the employed formulation limits the combined sample positions from $t+1$ to $t+W-1$ for a given window size W. Nevertheless, the number of correlation calculations is still large, and it is the most time-consuming to sum up $h_{i,\hat{k}}w_{i,t1}w_{i,t2}$, where $t1$ and $t2$ are the combined sample positions. Thus, we also accelerate this part with GPUs.

B. Optimization for Many-Core CPUs

The trace data is stored as a 2-D array, where the first dimension is the number of traces and the second dimension is the number of samples. The straightforward way to compute the correlation coefficients is to access the whole 2-D array for each key candidate. However, it might degrade the cache hit rate, especially when the number of traces is large. To enhance performance by accessing the trace data in a cache-friendly manner, we employ the loop tiling technique, which divides the loop into smaller nested loops. The traces are already partitioned for the progressive formulation. Therefore, a loop associated with the sample position is divided into two loops, one for the sample blocks and the other for the sample points in the block.

C. Leveraging Local Memory for GPUs

1) SO-CPA With Shared Memory: The kernel code for CPA is designed to assign each GPU thread the task of computing the summation of $h_{i,\hat{k}}w_{i,t}$ for a specific combination of i, $\hat{k}$, and t. In contrast, the kernel code for SO-CPA needs performance tuning using shared memory because the summation of $h_{i,\hat{k}}w_{i,t1}w_{i,t2}$ requires twice the number of memory accesses for sample points. Our preliminary experiments demonstrated that leveraging local memory (shared memory in CUDA) improves the performance of the SO-CPA.

In GPU programming, threads are organized hierarchically into grids and thread blocks, where each thread block contains multiple threads. Threads within a block can efficiently share data using local memory, which has significantly lower latency compared to global memory. As illustrated in Fig. 4, the trace data are partitioned into chunks, with each thread block

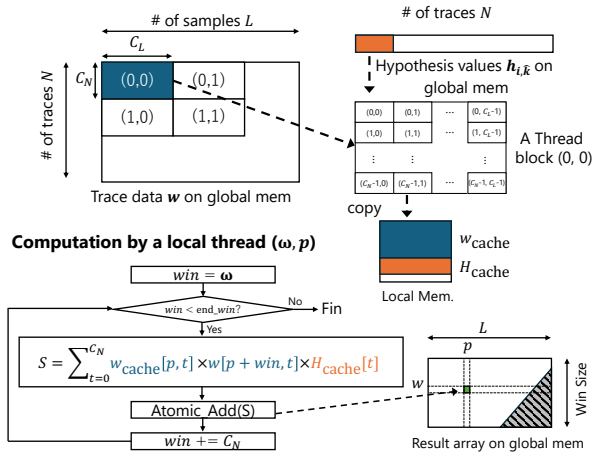


Fig. 4: SO-CPA with shared memory

assigned to process a single chunk. Each chunk is organized as a 2-D array (C_L, C_N), where C_N represents the number of traces and C_L corresponds to the number of samples in the block. Threads are also organized into the same 2-D shape. Its shape is fixed to (16, 32) in our implementation as a standard practice for GPU programming. First, each thread in the block loads a sample value and a hypothetical leakage value into the local memory. After synchronization, each thread (ω, p) calculates the partial summation of $h_{i,\hat{k}} w_{i,p} w_{i,p+\omega}$ for its assigned chunk, accessing the local memory. To compute the summation for all points within the window W, each thread iteratively computes the summation by incrementing the combined position as $\omega \leftarrow \omega + C_N$. Note that the combined sample $w_{i,p+\omega}$ is possibly out of the chunk. That is why it is always loaded from the global memory. The partial summation in each block is reduced to the final summation using the GPU's atomic operation feature.

D. Precision Consideration and Optimization

Although the progressive calculation of the correlation coefficient [see (4)] reduces the memory access count, it poses another issue regarding numerical precision. Typical oscilloscopes sometimes save the acquired power traces as integer values, such as 8-bit or 16-bit integers. However, the analyzed waveforms are often converted to floating-point representations. For example, averaged waveforms reduce noise, resulting in floating-point values [2]. In addition, several preprocessing techniques have been proposed to improve the performance of the CPA attack, which makes the waveforms floating-point values [37], [38]. Our preliminary study observed that the correlation coefficient for wide-ranging samples sometimes causes catastrophic cancellation. This is because the squared sum of the sample values and the sum of the squared sample values can be very close, and the subtraction between them may become zero due to rounding errors. In this case, the subtraction between them may become zero due to rounding errors. Although reducing the number of rounding operations using fused–multiply–add (FMA) can mitigate this issue, it remains unclear whether this technique avoids the problem in every case. The original implementation of ChipWhisperer, therefore, uses `numpy.longdouble` for the correlation calculation.

In contrast, we employ such an extended precision only for the most critical part of the correlation calculation to reduce the computational cost while ensuring compatibility with the original implementation. The other parts of the correlation calculation are implemented with double precision. However, the actual precision of this type depends on the platform and the compiler. For example, the precision of the `long double` is 80 bits mantissa on x86-64 with GCC by default, while it is equivalent to `double` on AppleSilicon ARM64 with Clang.

To realize quadruple precision (float128) or the extended precision of 80-bit mantissa without hardware support, we employ the software emulation technique [39]. A pair of float64 variables can be treated as a single float128 variable, and arithmetic operations between float128 values can be emulated with multiple float64 operations. This calculation part is out of the most time-consuming nested loops mentioned in Section IV-A. Therefore, the performance penalty due to the emulation is negligible.

Similarly, we can emulate double precision with two single precision variables. It helps GPUs to perform the correlation calculation efficiently. Even if the GPU supports double precision, the performance is often significantly lower than that of single precision for consumer GPUs. For example, the performance gap between single and double precision is 1:64 for NVIDIA RTX 4070 and 1:32 for AMD Radeon RX 7950 XTX. Thus, it is expected that the performance gains due to the hardware capability exceed the overhead introduced by the emulation.

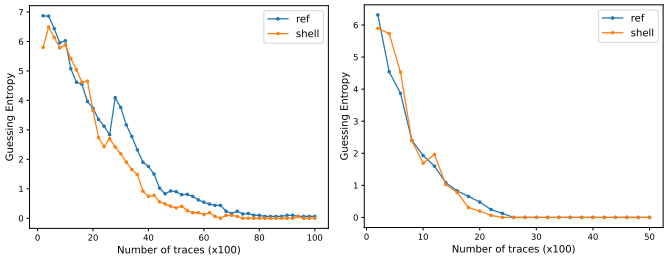
V. Experiment

In Section V-A, we validate that our FPGA shells do not impact leakage assessment results. Section V-B investigates optimal measurement settings for side-channel analysis instruments. Based on these findings, we conduct two case studies in Sections V-C and V-D. Finally, Section V-E benchmarks the performance of our CPA plugins using datasets from the case studies alongside publicly available datasets.

A. Validation of Leakage Assessment With the Shells

We carried out CPA attacks on the SAKURA-X and CW305 boards with and without the shells. The attack targets for both cases are hardware implementations of AES-128. For the SAKURA-X board, we used a factory pre-programmed design developed by AIST [26] as a reference implementation. For the CW305 board, we used an example design, an RTL of Project Google Vault, included in the ChipWhisperer repository as a reference implementation. We created wrappers for both RTL cores to make them compatible with our shells. To ensure a fair comparison, the shell-based designs were configured to operate at the same clock frequencies as their respective reference designs: 3 MHz for the SAKURA-X board and 10 MHz for the CW305 board. All the FPGA designs were synthesized and implemented using the Vivado Design Suite (version 2023.2). Those implementations are known to recover the key with a leakage model based on the Hamming distance between the last two round states [40].

To capture the power consumption waveform, we used Keysight InfiniVision MSO-X 4101A oscilloscopes with a 5



(a) SAKURA-X (b) CW305

Fig. 5: Guessing entropy comparison of shell-based and reference designs from [16], [26]

GSa/s sampling rate for the SAKURA-X board. To amplify the raw power consumption signal, we used a Langer EMV PA303 preamplifier with a 30 dB gain. For the CW305 board, we used ChipWhisperer-Lite with a 100 MSa/s sampling rate and a 30 dB gain for an amplifier inside the ChipWhisperer-Lite board.

To evaluate the leakage assessment results, we utilized guessing entropy (GE) [27] as the primary metric. Partial GE (PGE) for each key byte represents the ranking of the correct key among all possible candidate keys. If it is zero, it indicates that the attacker can identify the correct key byte. The GE is computed as the average of the PGE values across all key bytes.

Following the approach in [27], Fig. 5 shows logarithmic GE values for the SAKURA-X and CW305 boards. The GE values are plotted every 200 traces. In both cases, the tendency of the GE values is similar between the shell-based designs and the reference designs, and the shell-based designs converge to zero with slightly fewer traces than the reference designs. This indicates that the shells do not affect the leakage assessment. Hereinafter, we use the shell-based designs for the evaluation.

B. Explore the Appropriate Measurement Setting

Our FPGA shells do not have any limitation on the operational frequency for the cryptographic modules because the AXI bus protocol easily handles the clock domain crossing. Thus, we explore the appropriate measurement setting for side-channel analysis while changing the operational frequency of the cryptographic modules and the sampling rate of the oscilloscope. Although a similar discussion is also done in [41], its evaluated hardware is not an FPGA board but a microcontroller. Furthermore, it is the first work to compare the two different FPGA boards with the same design as far as we know.

We used the Google Vault design in this experiment. The Kintex-7 FPGA violates the timing constraint of the design when the operational frequency is set to 200 MHz. Thus, we set the operational frequency to 150 MHz for the SAKURA-X board. Due to the lower performance of the Artix-7 FPGA on the CW305 board, it can meet the timing constraint of up to 100 MHz.

For the SAKURA-X board, we captured the power consumption waveforms using the same instrumentation setup as described in Section V-A while varying the sampling rate across 100 MSa/s, 125 MSa/s, 500 MSa/s, 1.0 GSa/s, 2.5 GSa/s, and 5.0 GSa/s to evaluate its impact on leakage assessment. Fig. 6(a) shows the number of traces required to recover all key bytes for each sampling rate and operational frequency. The maximum number of traces is limited to 50 000. In the figure, $\times$ indicates no key byte is recovered, and 50+ means some key bytes are recovered but not all. The results suggest that the sampling rate of 2.5 GSa/s is at least needed for appropriate leakage assessment. However, the operational frequency does not significantly affect the leakage assessment results.

Upon further analysis of the waveforms, we identified spikes attributed to register writing. Interestingly, the spike width remains consistent regardless of the operational frequency. To effectively analyze these spikes, capturing approximately 20 sample points within an 8.0 ns window is sufficient. However, the ChipWhisperer-Lite board, with its maximum sampling rate of 100 MSa/s, is unable to capture such a rapid change.

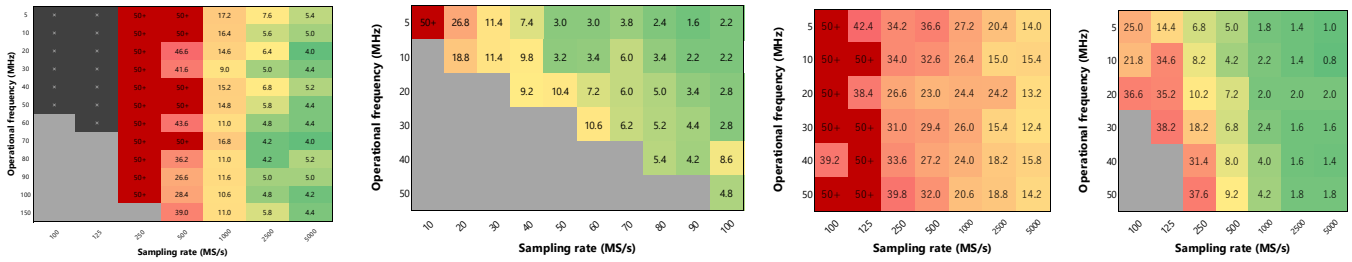
Similarly, we evaluated the CW305 board with two different capturing instruments: ChipWhisperer-Lite and Keysight MSO-X 4104A. For the ChipWhisperer-Lite, we set the sampling rate from 5 to 100 MSa/s with 10 MSa/s increments. Fig. 6(b) compares the number of traces required to recover all key bytes. ChipWhisperer-Lite shows a significantly better analysis performance than the MSO-X 4104A. For example, the ChipWhisperer-Lite can recover all key bytes only with 2200 traces when the sampling rate is set to 100 MSa/s and the operational frequency is 5 MHz.

On the other hand, the MSO-X 4104A requires 14 000 traces, even with much higher sampling rates of 5 GSa/s. As highlighted in [41], the ChipWhisperer-Lite is specifically designed for side-channel analysis, featuring an integrated low-noise amplifier that significantly enhances the quality of captured power consumption waveforms. In contrast, the MSO-X 4104A is a general-purpose oscilloscope, making it more susceptible to noise in the power consumption waveforms. This difference in design and purpose leads to the observed degradation in analysis performance when using the MSO-X 4104A.

To mitigate the noise effect on the analysis performance, we applied an FIR filter to the waveforms of the MSO-X 4104A. The tap number of the FIR filter is set to 2% of the waveform length, and the cutoff frequency is set to twice the operational frequency. The results are also shown in Fig. 6(b). As expected, the analysis performance is greatly improved by the FIR filter. Similar to the SAKURA-X board, a sampling rate of 2.5 GSa/s or higher ensures consistent analysis performance, regardless of the operational frequency. Nevertheless, with the same sampling rate of 100 MSa/s, the ChipWhisperer-Lite still outperforms the MSO-X 4104A with the FIR filter.

For the ChipWhisperer-Lite, the analysis results of 5 MHz become worse with a 40 MSa/s sampling rate or lower. Thus, at least eight samples (0.15 μs $\times$ 50 MS/s = 7.5 samples) per cycle are needed for the appropriate leakage assessment. That is why the ChipWhisperer-Lite also degrades the analysis performance for 40 MHz or higher operational frequency.

According to the results of this section, we suggest the following measurement settings for side-channel analysis.



(a) SAKURA-X (b) CW305

(i) ChipWhisperer-Lite (ii) MSO-X 4104A (iii) MSO-X 4104A+FIR

Fig. 6: Number of traces (in thousands) required to recover all key bytes under various conditions

Code 1: Evaluated remasking implementations in C language

(a) naive

```
m_x = mask->m_x;
m_ = mask->m_;
mask_val = m_x ^ m_;
for (i = 0; i < 4; i++) {
  state[i] ^= mask_val;
}
```

(b) separate XORing

```
m_x = mask->m_x;
for (i = 0; i < 4; i++) {
  state[i] ^= m_x;
}
m_ = mask->m_;
for (i = 0; i < 4; i++) {
  state[i] ^= m_;
}
```

(c) dummy load insertion

```
m_x = mask->m_x;
for (i = 0; i < 4; i++) {
  state[i] ^= m_x;
}
volatile uint32_t dummy;
*(volatile uint32_t*)&dummy;
__asm__ volatile ("":::"memory");
m_ = mask->m_;
for (i = 0; i < 4; i++) {
  state[i] ^= m_;
}
```

Note that m_x is m_x and m_ is a bundled m', i.e., $\{m',m',m',m'\}$.

- The sampling rate should be at least 2.5 GSa/s with the general-purpose oscilloscopes,
- The captured waveforms should be preprocessed with an FIR filter to reduce noise, mainly when the general-purpose oscilloscopes are used for the CW305 board, and
- In the case of ChipWhisperer-Lite, 30 MHz or lower operational frequency is recommended with a 100 MSa/s sampling rate.

In addition, as proposed in [41], clock synchronization between the FPGA and the ChipWhisperer-Lite probably improves the analysis performance. In the following case studies, we use the ChipWhisperer-Lite with a sampling rate of 100 MSa/s and an operational frequency of 5 MHz for the CW305 board.

C. Case Study: Compiler Optimization

Software implementations of cryptographic algorithms are still widely used in embedded systems. The software masking of AES-128 is a well-known countermeasure against SCAs. We evaluate the effect of compiler optimization on those implementations. While high-level languages like C enhance productivity and portability and leverage advanced optimizations from modern compilers, they may unintentionally introduce side-channel vulnerabilities. That is why some masking implementations are written in assembly language. Through the evaluation cycle, like Fig. 1, this section explores the secure implementation of AES-128 with the first-order masking in the C language without turning off compiler optimizations.

For Boolean masking [2], the S-box function is modified as $S_m(x) = S(x \oplus m) \oplus m'$, where m and m' are two random masks, and $S(x)$ represents the original S-box function. Additionally, four mask values, $m_x = \{m_1, m_2, m_3, m_4\}$, are generated. As a preprocessing step, these masks are passed through the MixColumns operation, resulting in the output values $m'_x = \{m'_1, m'_2, m'_3, m'_4\}$.

The plaintext is initially XORed with the mask m'_x, ensuring that the input to the encryption process is masked. Simultaneously, the round key is XORed with the combined mask $m'_x \oplus m$. As a result, each byte of the first round state is XORed with m, thereby canceling out m'_x. After the S-box and ShiftRows transformations, the mask m' is XORed to the state. Before the MixColumns operation, a critical step called *remasking* is executed to convert the mask m' into the mask m_x. This process guarantees that the masked values are consistently and securely propagated through all encryption rounds. In the final round, the mask m' is XORed with the final state to unmask it, producing the correctly encrypted output.

We implemented the masking AES-128 in C language and evaluated it on the RISC-V SoC with the CW305 shell. We used the C compiler GCC 14.2.0 based on the RISC-V GNU toolchain and Clang 19.1.7 with O2 optimization level. As a result of our preliminary evaluation, a naive implementation of the remasking process, which is shown in Code 1(a), is vulnerable to the first-order CPA attack. Fig. 7(a) and ((d)) shows the CPA results of the naive implementation with Clang and GCC, respectively. In addition to the PGE, we also present the success rate, which is defined as the proportion of correctly recovered key bytes to the total number of key bytes. A key byte is considered leaked if the PGE remains below 16 for the last ten evaluation steps.

The naive implementation [Code 1(a)] performs the mask conversion with a single XOR operation. The single XOR operation is designed to compute the four bytes of the state in parallel, leveraging the 32-bit processor architecture. In both cases (Clang and GCC), key bytes 0, 5, 10, and 15 are commonly leaked. Those key byte positions correspond to the

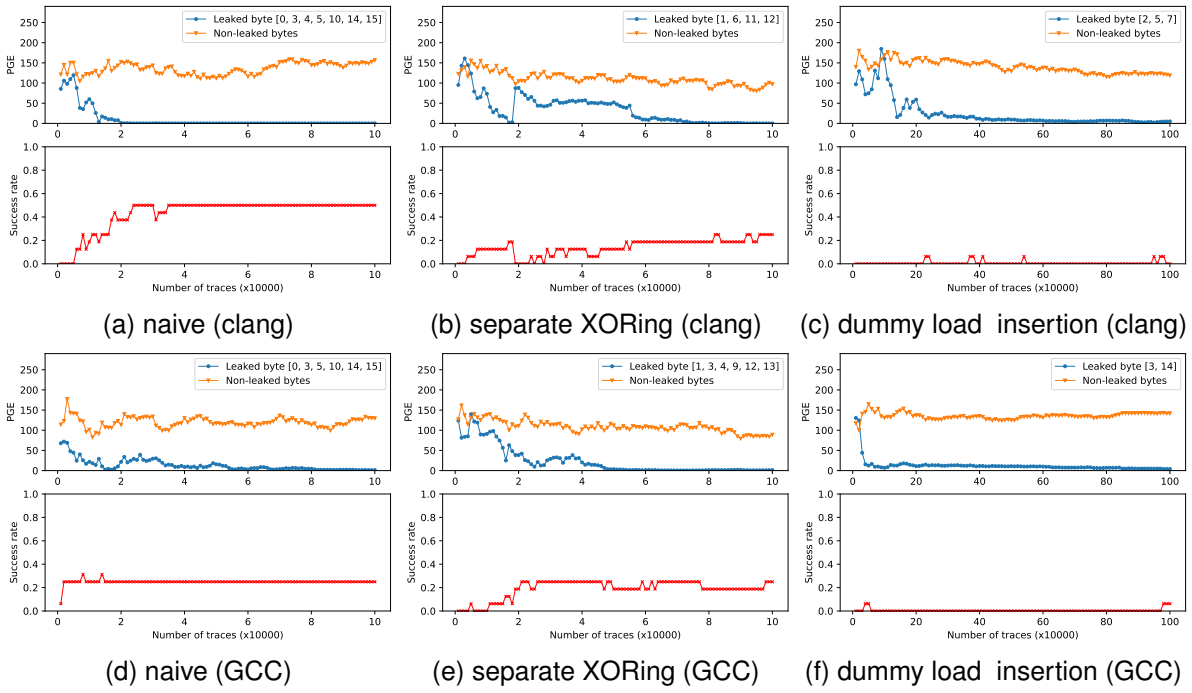


Fig. 7: CPA results of the remasking implementations in C language with compiler optimization

first four bytes of the state after the ShiftRows transformation. In general, compiler optimization tries to leverage the register file to reduce the number of memory accesses. This approach, however, introduces potential leakages due to microarchitectural effects, such as transitions within the same register or between pipeline registers [42]. Specifically, the first four bytes are likely loaded into the register file or pipeline registers and subsequently overwritten with the mask value m'. As a result, the Hamming distance of these transitions equals the Hamming weight of the unmasked state, resulting in a leakage that the CPA attack can exploit.

To avoid this leakage, we modified the remasking process as shown in Code 1(b), where the XOR operations for the mask m_x and m' are separated. By XORing the mask m_x first, the state is masked with two different masks, m_x and m'. Then, the mask m' is XORed to complete the remasking process. The results of the separate XORing implementation are shown in Fig. 7(b) and ((e)). Although those implementations successfully prevent the leakage of the first four key bytes, the last four key bytes (i.e., key bytes 12, 1, 6, 11) are likely leaked, especially with GCC. This leakage arises from another type of micro-architectural vulnerability known as memory remnant leakage [42]. It occurs due to transitions within the memory bus caused by successive memory accesses, even for different memory addresses. During the final iteration of the upper loop, the memory bus accesses the last four bytes of the state. Subsequently, the mask m' is fetched from memory. This sequence of memory operations likely causes transitions on the memory bus that correlate with the Hamming weight of the last four bytes of the unmasked state, thereby introducing a potential leakage.

Finally, we introduce a dummy load insertion to the remasking process to prevent memory remnant leakage, as shown in Code 1(c). The dummy load fetches a value from stack memory into a register. Additionally, a memory barrier instruction is included to enforce the memory access order. To demonstrate the resistance against the first-order CPA attack, we extended the captured traces to 1 000 000 and evaluated the dummy load insertion implementation. The results are shown in Fig. 7(c) and ((f)). Some bytes are still leaky, but the success rate is almost zero, and stable identification of the key bytes is not observed.

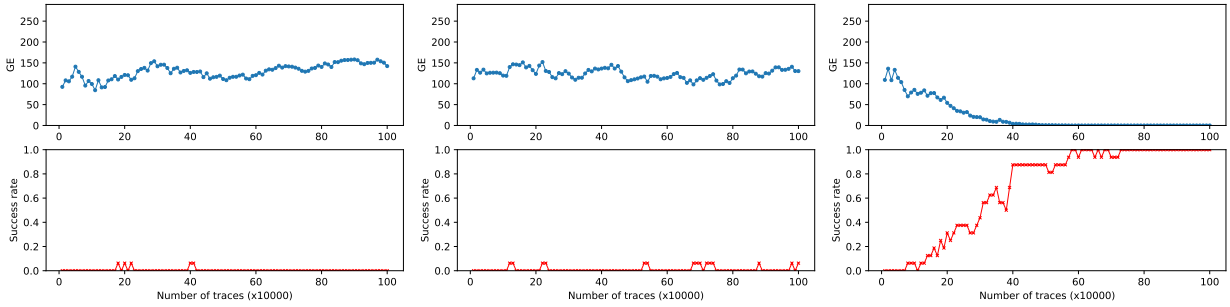
TABLE II: Performance comparison of the software implementations (cycles)

Implementation	Clang	GCC
Unprotected	2,668	4,055
Masking (naive)	5,111	6,501
No optimization for remasking	6,125 (+19.8 %)	7,571 (+16.4 %)
Masking (dummy load)	5,543 (+8.45 %)	6,695 (+2.98 %)

Table II summarizes the performance of the software implementations. The cycles to execute the AES-128 encryption on a single block are obtained by the performance counter of the control and status register (CSR) of the RISC-V SoC. Due to the updating of the new S-box table with flesh mask values, the masking implementations are slower than the unprotected implementation. However, the unprotected implementation can leak all key bytes with 500 traces. We can turn off compiler optimization only for the remasking process to prevent unexpected leakage. However, it causes performance degradation of 19.8% and 16.4% for Clang and GCC, respectively, compared to the naive implementation. Instead of such a conservative implementation, our final implementation with the dummy load insertion limits the performance overhead to 8.45% and 2.98%, respectively.

D. Case Study: HLS-Based Design

Like the compiler optimization for software implementations, we also evaluate the effect of the HLS optimization on



(a) RTL Implementation (b) HLS Implementation (c) HLS Implementation (3-stage)

Fig. 8: Comparison of RSM implementations

the hardware implementations. As a case study, we employ AMD Vitis HLS (version 2023.2) to implement the rotating S-Box masking (RSM) countermeasure [43] for AES-128. In addition to the HLS implementation, we also implemented the RSM in RTL for comparison. The HLS tool automatically applies pipelining to the design when a higher target frequency is specified. However, pipelined sensitive operations can potentially introduce new leakage points. To evaluate this, we synthesized the HLS design with two different target frequencies: 20 and 100 MHz. The 20 MHz design is synthesized without pipelining, while the 100 MHz design is synthesized with a three-stage pipeline for the AES round. This comparison allows us to assess the impact of pipelining.

Fig. 8 shows the comparison of the RSM implementations. The leakage model is the last round state Hamming distance, the same as in Section V-A. Both the RTL and nonpipelined HLS implementations demonstrate resistance to first-order CPA attacks, as expected. However, the pipelined HLS implementation exhibits vulnerabilities, with all key bytes being identified using approximately 800 000 traces. We also conducted a similar evaluation on a four-stage pipelined HLS design, which yielded comparable leakage results.

In the pipelined design, the first stage is responsible for the masked S-box operation, the second stage handles the ShiftRows transformation, and the last stage performs the remaining operations, including the remasking process. According to the CPA results, we located the leakage point in the last stage. The output of the last stage is stored in the state registers before being fed back to the first stage for the next AES round. However, the same transition of state registers occurs even in the nonpipelined design without causing any leakage. The masked S-box is realized using Block RAMs constructed from SRAM macros. It is well known that the dynamic power consumed during SRAM read operations—primarily due to the charging and discharging of bitlines—is larger than that caused by bit transitions in flipflops. The first stage, which involves SRAM access, consumes significantly more power than the subsequent stages. Although potential leakage exists even in a nonpipelined design, the power consumption associated with SRAM reads serves as noise for CPA attacks. Nevertheless, in the pipelined design, this noise component is temporally separated from the state register updates due to the pipeline stages. To mitigate this leakage and develop anti-tamper HLS tools, another mask should be fed into the predecessor stage even when there is no valid data to process due to loop-carried dependencies.

TABLE III: Testbeds for the CPA and SO-CPA acceleration

System	CPU (# of Cores/Threads)	GPU
No. 1	Ryzen 9 7950X (16C/32T)	NVIDIA RTX 4070
No. 2	Intel Core i9-14900KF (24C/32T)	Radeon RX 7900 XTX
No. 3	Ryzen7 5700G (8C/16T)	Intel Arc A770
No. 4	Apple M1 Ultra (20C)	Integrated 64-core GPU

TABLE IV: Benchmarks and conditions for the evaluation

First-order CPA			
Dataset	# of traces	# of samples	Win. size
DPA Contest v2 [44]	20,000	3,253	-
RV-Mask in Sec. V-C	200,000	11,800	-
Second-order CPA			
ASCAD [45]	10,000	700	300
RV-Mask in Sec. V-C	40,000	3000	1000

E. CPA and SO-CPA Acceleration

Lastly, we compare the acceleration performance of our CPA and SO-CPA plugins with other frameworks described in Section II-C. The evaluation is conducted on four testbeds, as summarized in Table III. The first three testbeds operate on Linux OS, with all tools executed within Docker containers to maintain consistent library and compiler versions across evaluations. We used Python 3.12.3 and GCC 13.3.0 as the compiler within an Ubuntu 24.04 base image. For NVIDIA GPUs, driver version 570.144 and CUDA toolkit version 12.8 were utilized. For AMD GPUs, ROCm 6.3.4 served as the driver and OpenCL runtime. For Intel GPUs, the Intel graphics compute runtime version 25.09.32961.8 was employed. The fourth testbed operates on macOS 15.4.1, where our framework runs natively without Docker. We used Python 3.12.6 and Apple Clang 17.0.0 in Xcode 16.3.

We utilize two datasets for the first-order CPA, as detailed in Table IV. The first dataset is the publicly available dataset [44], from which we extracted 20 000 traces corresponding to a fixed key (6a 51 a0 d2 d8 54 2f 68 96 0f a7 28 ab 51 33 a3 is selected). The second dataset consists of 200 000 traces collected during the experiments in Section V-C (our final implementation with Clang).

Likewise, for the SO-CPA, we used two additional datasets, as detailed in Table IV. The first dataset is the publicly avail-

TABLE V: Performance comparison of the CPA

		DareDevil [21]	RamDPA [20]	SICAK [24]		Our framework					
						OpenMP		CUDA		OpenCL	
				OpenMP	OpenCL	w/ tiling	w/o tiling	FP64	FP32	FP64	FP32
DPA Contest v2	No. 1	6.11	47.48	22.08	6.75	5.95	**5.60**	2.83	**1.30**	2.89	1.44
	No. 2	6.98	35.82	18.34	5.19	7.34	**5.50**	-	-	2.93	**1.72**
	No. 3	13.46	137.00	26.55	n/a	**10.75**	44.11	-	-	n/a	**6.86**
	No. 4	16.41	41.61	n/a	n/a	**6.28**	17.33	-	-	n/a	**1.91**
RV-Mask	No. 1	397.78	1719.40	701.01	213.95	228.37	**217.24**	88.37	34.58	88.97	**34.56**
	No. 2	286.98	1316.29	490.55	101.06	349.31	**252.81**	-	-	47.22	**45.94**
	No. 3	732.91	5019.95	2239.14	n/a	**689.48**	1296.02	-	-	n/a	**254.88**
	No. 4	607.89	1498.12	n/a	n/a	**262.90**	752.15	-	-	n/a	**87.75**

TABLE VI: Performance comparison of the SO-CPA

		DareDevil [21]	Our framework								
			OpenMP	CUDA				OpenCL			
				w/ LM		w/o LM		w/ LM		w/o LM	
				FP64	FP32	FP64	FP32	FP64	FP32	FP64	FP32
ASCAD	No. 1	73.46	**69.52**	59.92	32.55	101.48	32.72	78.70	43.90	101.40	**31.93**
	No. 2	**73.65**	77.01	-	-	-	-	40.16	38.79	51.85	**31.77**
	No. 3	**198.10**	226.69	-	-	-	-	n/a	n/a	n/a	256.11
	No. 4	148.33	**88.25**	-	-	-	-	n/a/	n/a	n/a	**52.05**
RV-Mask	No. 1	4377.43	**4248.23**	3402.07	**1582.76**	5459.88	1686.88	3882.62	2004.90	5442.48	1653.61
	No. 2	4397.15	**4369.05**	-	-	-	-	1937.06	**1768.14**	2831.79	1890.60
	No. 3	**12771.35**	14100.96	-	-	-	-	n/a	n/a	n/a	21843.93
	No. 4	8512.72	**4885.30**	-	-	-	-	n/a	n/a	n/a	5777.03

able ASCAD dataset [45], from which we extracted the first 10 000 traces. This is sufficient to recover the key. Although the target 700 samples are related to the third key byte, we did not restrict the attacked key bytes only to the third byte for this evaluation. The second dataset is derived from the same experiment as the first-order CPA, but we limited the number of traces to 40 000 and the sample points to 3000 to manage the fairly long execution time of the SO-CPA. This dataset is sufficient to recover a single key byte, although recovering all key bytes would require approximately 150 000 traces.

For the comparison, we utilized the following existing frameworks: DareDevil [21] (commit 81cb57f), RamDPA [20] (commit 28ad3b0), and SICAK [24] (release 1.2.1, applicable only for Linux OS). To ensure accurate performance measurement, the gnuplot integration in the main process of RamDPA, which is used for result plotting, was commented out in the source code. Additionally, SICAK comprises multiple separate tools, and the reported time focuses solely on the correlation calculation, excluding the hypothesis calculation phase.

Table V gives the performance comparison of the first-order CPA. Note that DareDevil cannot recover any key byte for the DPA Contest v2 dataset since it does not support any Hamming-distance model. Our framework provides two flavors of the OpenMP implementation, which are with and without tiling. Even though DareDevil is a sophisticated framework and takes a much shorter time than RamDPA and SICAK, our OpenMP implementation outperforms it in all testbeds. Notably, the Apple Silicon CPU (No. 4) demonstrates a significant performance boost with our OpenMP implementation, achieving more than double the speed of DareDevil. However, the tiling is not always effective. For example, testbeds No. 1 and No. 2 show better performance with tiling, while the other two testbeds show better performance without tiling. As a result of additional experiments with testbed No. 2, we found that the tiling is effective when the number of threads is larger than 8. Thus, we conclude that the tiling is effective for the CPUs with a large number of cores, such as No. 1 and No. 2.

The GPU implementations of our framework achieve further performance improvements over the OpenMP implementation. Even with double-precision floating-point operations (FP64), our CUDA and OpenCL implementations outperform the SICAK OpenCL implementation. Note that its implementation relies on the native double-precision floating-point operations so that it is not available for Intel GPUs (No. 3). By leveraging double-precision emulation using single-precision operations (FP32), the CUDA and OpenCL implementations reduce execution time, especially for the NVIDIA GPU (No. 1). Compared to the state-of-the-art tool, DareDevil, our OpenMP implementation delivers performance improvements ranging from 1.09× to 2.62× for the DPA Contest v2 dataset and 1.06× to 2.31× for the RV-Mask dataset. When leveraging GPU implementations, the performance gains are even more pronounced, achieving 1.96× to 8.60× for the DPA Contest v2 dataset and 2.20× to 6.93× for the RV-Mask dataset. We also profiled the CUDA implementation for the DPA Contest v2 dataset on testbed No. 1 to analyze the performance bottleneck. The profiling results indicate that the correlation computation kernel accounts for approximately 85% of the total execution time, while data transfer between the host and device consumes around 2.2%. This result suggests that memory access is not the primary performance bottleneck,

implying that further acceleration can be achieved using a more powerful GPU.

As for the SO-CPA, we compare the performance of our framework only with DareDevil since HO-CPA in SICAK is based on a different formulation. Table VI summarizes the performance comparison of the SO-CPA. Testbed No. 3 exhibits slower performance compared to the other testbeds and DareDevil. This is likely due to the limited size of the last-level cache, which is only 16 MB for the Ryzen 7 5700G, significantly smaller than the other testbeds (36–64 MB). Additionally, the Intel Arc A770 GPU (No. 3) cannot fully leverage the local-memory-enabled implementation due to static memory allocation in the OpenCL kernel. Specifically, the required local memory for all threads exceeds the available local memory size of the GPU. Combined with the GPU's limited computational power, this results in the OpenCL implementation underperforming relative to the OpenMP implementation. The OpenMP implementation for testbed No. 2 is also slower than DareDevil for the ASCAD dataset, whereas it outperforms DareDevil for the RV-Mask dataset. Thus, it is comparable to DareDevil in terms of performance. For the remaining testbeds (No. 1 and No. 4), our OpenMP implementation achieves a performance improvement over DareDevil. Especially for testbed No. 4, 68.1% and 74.3% performance improvements are achieved for the ASCAD and RV-Mask datasets, respectively.

Our GPU implementations demonstrate modest acceleration for the SO-CPA, except for testbeds No. 3 and No. 4. Among the GPU variants, the local-memory-enabled implementation (w/ LM) proves effective for the FP64 version. In contrast, the nonlocal-memory-enabled implementation (w/o LM) is generally more efficient for the FP32 version. On the Apple Silicon GPU (No. 4), the lack of support for 64-bit atomic operations limits the applicability of the local-memory-enabled implementation, resulting in performance improvements only for smaller datasets like ASCAD. Nevertheless, for mid-range or high-end GPUs, selecting the appropriate GPU implementation variant yields substantial performance gains over DareDevil, achieving up to 2.3× acceleration for the ASCAD dataset and 2.77× for the RV-Mask dataset.

VI. Conclusion

Side-channel vulnerabilities in cryptographic implementations represent a critical challenge to the security of hardware roots of trust. Designers must ensure that their modules are resilient to such attacks prior to deployment. In this article, we introduced an open-source framework designed to streamline and enhance the efficiency of side-channel analysis for cryptographic implementations. By utilizing the FPGA shells provided within the framework, we conducted comprehensive experiments to determine the optimal configurations for two widely used FPGA boards in the side-channel analysis community: SAKURA-X and CW305. These insights will assist designers in preventing the use of improper methods when evaluating side-channel vulnerabilities. Furthermore, we revealed that compiler optimizations can potentially increase side-channel leakage, and we explored mitigation strategies through a case study. Finally, we demonstrated that our many-core and GPU-accelerated CPA plugins significantly reduce the time required for CPA evaluations, achieving up to an 8.60× speedup over state-of-the-art CPA tools. For SO-CPA, although the acceleration is more limited compared to first-order CPA, our plugins still achieve up to a 2.77× speedup using GPUs. Currently, our framework supports two famous FPGA boards, SAKURA-X and CW305. In future work, we extend the software library to support Zynq series FPGAs, which inherently support the AXI bus. In addition, we plan to extend power measurement capabilities by incorporating adaptive sampling and online leakage detection techniques to reduce evaluation time further. Using our framework, we will continue to investigate side-channel vulnerabilities in various cryptographic implementations and other countermeasure implementations with HLS tools.

References

[1] A. A. Zharkova, "Application Encryption with Trusted Platform Module to Implement Standards in Windows 11 Environment," in *2023 Seminar on Information Computing and Processing (ICP)*. IEEE, 2023, pp. 239–241.

[2] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer Science & Business Media, 2008, vol. 31.

[3] T. B. Singha, R. P. Palathinkal, and S. R. Ahamed, "Securing AES designs against power analysis attacks: A survey," *IEEE Internet of Things Journal*, vol. 10, no. 16, pp. 14 332–14 356, 2023.

[4] Common Criteria, "Application of Attack Potential to Smartcards and Similar Devices, Version 3.2.1," https://commoncriteriaportal.org/files/ccfiles/JIL-Application-of-Attack-Potential-to-Smartcards-v3.2.1.pdf, February 2024.

[5] ISO 17825:2016(E), "Information technology - Security techniques - Testing methods for the mitigation of non-invasive attack classes against cryptographic modules," International Organization for Standardization, Geneva, CH, Tech. Rep., Mar. 2016.

[6] M. Azouaoui, D. Bellizia, I. Buhan, N. Debande, S. Duval, C. Giraud, É. Jaulmes, F. Koeune, E. Oswald, F.-X. Standaert *et al.*, "A systematic appraisal of side channel evaluation strategies," in *International Conference on Research in Security Standardisation*. Springer, 2020, pp. 46–66.

[7] Y. Wang and M. Tang, "A survey of side-channel leakage assessment," *Electronics*, vol. 12, no. 16, p. 3461, 2023.

[8] G. Becker, J. Cooper, E. DeMulder, G. Goodwill, J. Jaffe, G. Kenworthy, T. Kouzminov, A. Leiserson, M. Marson, P. Rohatgi *et al.*, "Test vector leakage assessment (TVLA) methodology in practice," in *International Cryptographic Module Conference*, vol. 1001. sn, 2013, p. 13.

[9] W. Unger, L. Babinkostova, M. Borowczak, R. Erbes, and A. Srinath, "TVLA, Correlation Power Analysis and Side-Channel Leakage Assessment Metrics," in *Proceedings of the Lightweight Cryptography Workshop*, vol. 42, 2022.

[10] M. Kar, A. Singh, S. K. Mathew, A. Rajan, V. De, and S. Mukhopadhyay, "Reducing power side-channel information leakage of AES engines using fully integrated inductive voltage regulator," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 8, pp. 2399–2414, 2018.

[11] A. P. Fournaris, A. Moschos, and N. Sklavos, "Side Channel Assessment Platforms and Tools for Ubiquitous Systems," *Security of Ubiquitous Computing Systems: Selected Topics*, pp. 147–163, 2021.

[12] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2004, pp. 16–29.

[13] E. Prouff, M. Rivain, and R. Bevan, "Statistical analysis of second order differential power analysis," *IEEE Transactions on computers*, vol. 58, no. 6, pp. 799–811, 2009.

[14] S. Picek, G. Perin, L. Mariot, L. Wu, and L. Batina, "Sok: Deep learning-based physical side-channel analysis," *ACM Computing Surveys*, vol. 55, no. 11, pp. 1–35, 2023.

[15] "SAKURA (SASEBO-GIII)," https://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-X.html, Accessed on 2024-05-13.

[16] C. O ' flynn and Z. Chen, "Chipwhisperer: An open-source platform for hardware embedded security research," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2014, pp. 243–260.

[17] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2002, pp. 13–28.

[18] B. Timon, "Non-profiled deep learning-based side-channel attacks with sensitivity analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 107–131, 2019.

[19] Q. L. Meunier, "FastCPA: Efficient correlation power analysis computation with a large number of traces," in *Proceedings of the Sixth Workshop on Cryptography and Security in Computing Systems*, 2019, pp. 7–12.

[20] A. F. Rodríguez, L. H. Encinas, A. M. Muñoz, and B. A. Alcázar, "A modular and optimized toolbox for side-channel analysis," *IEEE Access*, vol. 7, pp. 21 889–21 903, 2019.

[21] P. Bottinelli and J. W. Bos, "Computational aspects of correlation power analysis," *Journal of Cryptographic Engineering*, vol. 7, pp. 167–181, 2017.

[22] T. Swamy, N. Shah, P. Luo, Y. Fei, and D. Kaeli, "Scalable and efficient implementation of correlation power analysis using graphics processing units (GPUs)," in *Proceedings of the Third Workshop on Hardware and Architectural Support for Security and Privacy*, 2014, pp. 1–8.

[23] H. Gamaarachchi, R. Ragel, and D. Jayasinghe, "Accelerating correlation power analysis using graphics processing units (gpus)," in *7th International Conference on Information and Automation for Sustainability*. IEEE, 2014, pp. 1–6.

[24] P. Socha, V. Miskovsky, and M. Novotny, "SICAK: An open-source SIde-Channel Analysis toolKit," in *8th Workshop on Trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE 2019)*, vol. 5, 2019.

[25] E. Ferrufino, L. Beckwith, A. Abdulgadir, and J.-P. Kaps, "FOBOS 3: An Open-Source Platform for Side-Channel Analysis and Benchmarking," in *Proceedings of the 2023 Workshop on Attacks and Solutions in Hardware Security*, 2023, pp. 5–14.

[26] Y. Nomata, M. Matsubayashi, K. Sawada, and A. Satoh, "Comparison of side-channel attack on cryptographic cirucits between old and new technology fpgas," in *2016 IEEE 5th Global Conference on Consumer Electronics*. IEEE, 2016, pp. 1–4.

[27] K. Papagiannopoulos, O. Glamočanin, M. Azouaoui, D. Ros, F. Regazzoni, and M. Stojilović, "The side-channel metrics cheat sheet," *ACM Computing Surveys*, vol. 55, no. 10, pp. 1–38, 2023.

[28] E. Prouff, M. Rivain, and R. Bevan, "Statistical analysis of second order differential power analysis," *IEEE Transactions on computers*, vol. 58, no. 6, pp. 799–811, 2009.

[29] P. Socha, V. Miškovskỳ, and M. Novotnỳ, "First-order and higher-order power analysis: Computational approaches and aspects," in *2019 8th Mediterranean Conference on Embedded Computing (MECO)*. IEEE, 2019, pp. 1–5.

[30] C. Kenyon and C. Capano, "Apple silicon performance in scientific computing," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2022, pp. 1–10.

[31] M. H. Quraishi, E. B. Tavakoli, and F. Ren, "A survey of system architectures and techniques for fpga virtualization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 9, pp. 2216–2230, 2021.

[32] B.-A. Dao, T.-T. Hoang, A.-T. Le, A. Tsukamoto, K. Suzaki, and C.-K. Pham, "Correlation power analysis attack resisted cryptographic RISC-V SoC with random dynamic frequency scaling countermeasure," *IEEE Access*, vol. 9, pp. 151 993–152 014, 2021.

[33] B. Hettwer, K. Das, S. Leger, S. Gehrer, and T. Güneysu, "Lightweight side-channel protection using dynamic clock randomization," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2020, pp. 200–207.

[34] M. Karabulut and R. Azarderakhsh, "Efficient Cpa Attack on Hardware Implementation of Ml-Dsa in Post-Quantum Root of Trust," in *2025 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2025, pp. 392–402.

[35] M. Masaki, K. Takuya, I. Haruto, T. Hideki, and N. Hiroshi, "A Data Compression Method for DL-SCAs using Denoising Autoencoders," in *IEEE Symposium on Low-Power and High-Speed Chips and Systems (COOL Chips 28)*. IEEE, 2025, pp. 335–336.

[36] "VexRiscv: A FPGA friendly 32 bit RISC-V CPU implementation," https://github.com/SpinalHDL/VexRiscv, 2014–2025, accessed: 2025-05-19.

[37] T.-H. Tran, D.-T. Dam, B.-A. Dao, V.-P. Hoang, C.-K. Pham, and T.-T. Hoang, "Compacting side-channel measurements with amplitude peak location algorithm," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 32, no. 3, pp. 573–586, 2023.

[38] L. Wu and S. Picek, "Remove some noise: On pre-processing of side-channel measurements with autoencoders," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 389–415, 2020.

[39] A. Thall, "Extended-precision floating-point numbers for GPU computation," in *ACM SIGGRAPH 2006 research posters*, 2006, pp. 52–es.

[40] S. Picek, A. Heuser, A. Jovic, S. Bhasin, and F. Regazzoni, "The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 209–237, 2019.

[41] C. O'Flynn and Z. Chen, "Synchronous sampling and clock recovery of internal oscillators for side channel analysis and fault injection," *Journal of Cryptographic Engineering*, vol. 5, pp. 53–69, 2015.

[42] J. Zeitschner and A. Moradi, "PoMMES: Prevention of Micro-architectural Leakages in Masked Embedded Software," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2024, no. 3, pp. 342–376, 2024.

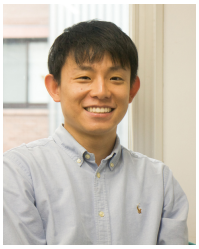
[43] M. Nassar, Y. Souissi, S. Guilley, and J.-L. Danger, "RSM: A small and fast countermeasure for AES, secure against 1st and 2nd-order zero-offset SCAs," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2012, pp. 1173–1178.

[44] C. Clavier, J.-L. Danger, G. Duc, M. A. Elaabid, B. Gérard, S. Guilley, A. Heuser, M. Kasper, Y. Li, V. Lomné *et al.*, "Practical improvements of side-channel attacks on AES: feedback from the 2nd DPA contest," *Journal of Cryptographic Engineering*, vol. 4, pp. 259–274, 2014.

[45] R. Benadjila, E. Prouff, R. Strullu, E. Cagli, and C. Dumas, "Deep learning for side-channel analysis and introduction to ASCAD database," *Journal of Cryptographic Engineering*, vol. 10, no. 2, pp. 163–188, 2020.



Takuya Kojima (Member, IEEE) received the Ph.D. degrees from Keio University in 2021. He was an assistant professor at The University of Tokyo from 2021 to 2025, and is currently an Associate Professor at University of Tsukuba. His research interests include computer architecture and compiler techniques, especially focusing on reconfigurable devices.



Masaki Morita received the B.E. degree from the University of Tokyo in 2024, where he is currently working toward the M.E. degree. His research interests include computer security, with a particular focus on hardware security, including secure computer architecture and hardware-based protection mechanisms.



Hideki Takase received his Ph.D. degree of Information Science from Nagoya University in 2012. He had been an Assistant Professor at Graduate School of Informatics, Kyoto University since 2012, and was promoted to an Associate Professor in 2019. He moved to Graduate School of Information Science and Technology, The University of Tokyo in 2021. His research interests include runtime platform and system-level design methodology for embedded/real-time/IoT computing.



Hiroshi Nakamura (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from the University of Tokyo, Tokyo, Japan, in 1990. He is currently a Professor and Dean of the Graduate School of Information Science and Technology, University of Tokyo. His research interests include high-performance computing, power-aware computing, secure computing, and quantum computing.