EvilCS: An Evaluation of Information Leakage through Context Switching on Security Enclaves

Aruna Jayasena, Richard Bachmann and Prabhat Mishra
University of Florida, Gainesville, Florida, USA

***Abstract*—Security enclaves provide isolated execution environments for trusted applications. However, modern processors utilize diverse performance enhancement methods (e.g., branch prediction and parallel execution) that can introduce security vulnerabilities. Specifically, if a processor leaks any information, an adversary can monitor and recover secrets from trusted applications. This paper makes a connection between context switching and information leakage in security enclaves. We present an evaluation framework that analyzes the potential channels through which context switching can expose sensitive information across the security enclave boundaries as a physical side-channel signature. Specifically, we propose a statistical information leakage assessment technique to evaluate the side-channel leakage of a security enclave during the pre-silicon design stage. Experimental evaluation on multiple RISC-V security enclaves reveals that context switching introduces power side channels that an adversary can exploit to infer the execution sequences as well as register values of trusted applications.**

I. Introduction

System-on-Chip (SoC) design needs to consider various conflicting constraints, including area, power, performance, and security. For example, modern processors utilize performance enhancement features (e.g., branch prediction and speculative execution) that can negatively impact area and power requirements. The power usage and performance are important considerations since it has a direct impact on energy efficiency and thermal regulation, which in turn affects the battery life. Previous studies have demonstrated that the power consumption of a processor can inadvertently leak sensitive information through the power and electromagnetic side channels [1], [2]. These side channels arise from the variations in power consumption caused by different internal operations, such as instruction execution and data access. An adversary can exploit these side-channel leaks to deduce valuable internal details about the processor's operations, potentially leading to the extraction of cryptographic keys, confidential data, or even details about the execution of specific instructions. Due to significant concerns regarding security and privacy, there has been a widespread adoption of security enclaves, often known as trusted execution environments (TEEs).

A. Security Enclave Kernels

The objective of a trusted execution environment is to maintain the confidentiality and integrity of sensitive applications by isolating them from other (potentially compromised) applications. Security enclaves provide process isolation utilizing the specific hardware functionalities while providing basic kernel functionalities such as scheduling, context switching, and interrupt handling. Open-source instruction sets such as RISC-V have enabled the development of processor designs customized to match specific application requirements. This adaptability allows for the formation of various security enclave setups, utilizing different hardware IP cores in conjunction with a range of security enclave kernels. Keystone [3], Multizone [4], and OpenMZ [5] are example security enclave kernels targeted at RISC-V instruction-set architecture. These security enclave kernels effectively utilize the physical memory protection (PMP) feature available in the RISC-V instruction set to provide process isolation. This allows the user to run untrusted applications alongside security-sensitive applications, without the former infringing on the latter.

This work was partially supported by the grants from NSF (CCF-1908131) and Semiconductor Research Corporation (2022-HW-3128).

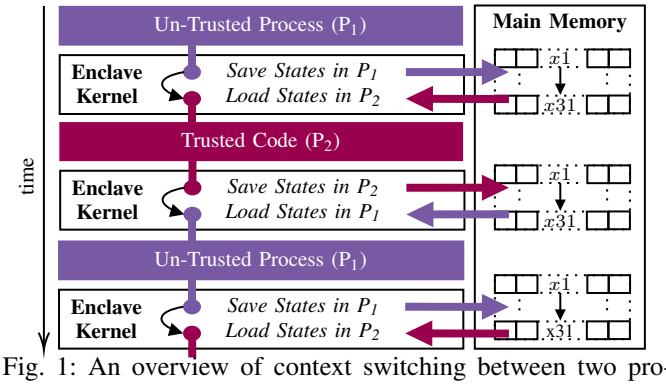


Fig. 1: An overview of context switching between two processes (P_1 and P_2). When switching from P_1 to P_2, the enclave kernel creates a hardware thread (Hart) and saves the context of P_1 (e.g., contents of the registers) into the memory. Similarly, it needs to load the contents of P_2 from the memory (restore context) into the registers before P2 starts execution.

B. Hardware Threads: Hart

Simultaneous multithreading (SMT) in RISC-V introduces the concept of "***Hart***" to enhance throughput and improve utilization in a processing core. In a single-core processor with hardware multithreading, a hart represents an independent thread of execution. Each *hart* operates as a virtualized execution unit within the same physical core. They share the same execution resources, such as the execution pipelines and functional units, while maintaining their separate sets of architectural registers and program counters. This allows multiple threads to make progress simultaneously within the same core. The process of isolating the *hart* is made possible through a context-switching call from the system kernel or the operating system. Figure 1 illustrates a scenario where a security enclave kernel is saving the general purpose registers (GPRs) and control and status registers (CSRs) during the execution of two

979-8-3503-4859-0/DATE24/© 2024 EDAA

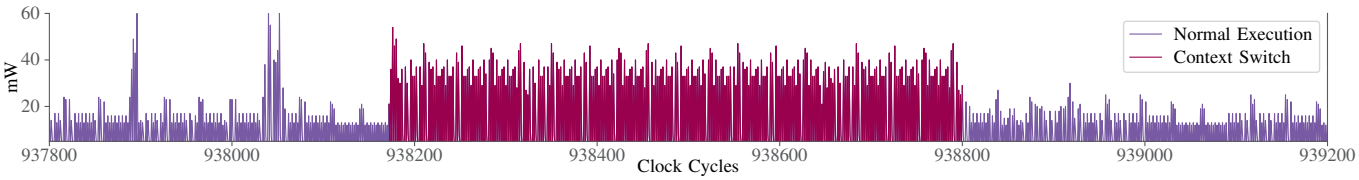


Fig. 2: Context switching power profile of a RISC-V SoC running OpenMZ Security Kernel compared with the normal execution. The context-switching process can be visually observed by a simple power analysis due to the block-wise change of system registers and utilization of the memory bus for bulk read/write transfer of register values.

processes of $\mathbf{P_1}$ and $\mathbf{P_2}$. Modern processors employ preemptive thread termination through timed interruptions to facilitate the context-switching process. This mechanism enables the kernel to regain control when necessary.

C. Threat Model: *EvilCS* Vulnerability

To start a context switch, the hardware needs to create an interrupt, pause the execution of the current process, and let the kernel take over the process handling. In return, the kernel needs to save register values from the previous (paused) execution process and load the register values from the new execution process. A common implementation of this kernel functionality is illustrated in Figure 3. This process involves bulk register value swaps and memory bus transfers to load and store values involved in the context-switching process. Both register value swaps and memory bus transfers significantly change the side-channel signature of the device [6]. Figure 2 illustrates a power side channel signature of a RISC-V-based SoC running OpenMZ security kernel during a context switch process. This creates a perfect opportunity for an adversary to analyze the side-channel signature to retrieve information. We refer this capability as "***EvilCS***", where an adversary maliciously utilizes the context switch to recover sensitive data from the trusted application in the security enclave. In order to exploit the EvilCS, the adversary should have physical access to the device to observe the power signature of the SoC. We assume that the kernel and hardware implementations are open source (e.g., RISC-V). Therefore, an adversary can identify the timing information to exploit EvilCS to launch an attack. The adversary is interested in recovering the GPR values rather than the CSR values since their objective is to recover computations within the security enclave and not the configuration of the enclave which may be publicly available.

D. Contributions

We propose a statistics-based strategy for assessing both the hardware and firmware of security enclave implementations. Specifically, this paper makes the following contributions,

- We show that sensitive register data of trusted applications running on an SoC can be recovered via power signature during the context switching.
- We propose a test generation technique to maximize the side-channel sensitivity of the context switch process.
- We formulate a change point detection technique that automatically isolates the power signature that is related to the context-switch process from the power profile.

```
.macro ctx_save base
    sw ra, 0(\base)
    sw sp, 4(\base)
    sw s0, 8(\base)
    sw s1, 12(\base)
    sw s2, 16(\base)
    sw s3, 20(\base)
    sw s4, 24(\base)
    ...
.endm
```

(a) GPR Store (`sw`) Macro

```
.macro ctx_load base
    lw ra, 0(\base)
    lw sp, 4(\base)
    lw s0, 8(\base)
    lw s1, 12(\base)
    lw s2, 16(\base)
    lw s3, 20(\base)
    lw s4, 24(\base)
    ...
.endm
```

(b) GPR Load (`lw`) Macro

```
.globl sys_switch
sys_switch:
    ctx_save a0   # a0 => struct context *old
    ctx_load a1   # a1 => struct context *new
    ret           # pc=ra; swtch to new task (new->ra)
```

(c) Register Swap Procedure

Fig. 3: Implementation of switching between two *harts* using GPR swap procedure in xv6-RISC-V kernel. This is the common implementation procedure used by security kernels.

- We implement a power analysis technique to evaluate the correlation between the register values and the power consumption of the device during context switch.
- We consider widely used security enclave kernels with different hardware implementations to form sixteen security enclave configurations and evaluate them for the existence of EvilCS vulnerability.

This paper is organized as follows. Section II surveys related efforts. Section III describes our statistical information leakage analysis framework. Section IV presents experimental results. Finally, Section V concludes the paper.

II. Related Work

Test Vector Leakage Assessment (TVLA) is a popular method for evaluating side-channel leakage of hardware implementations [7]. This process involves generating input test patterns that make distinguishable differences in the power signature of the device. The specific TVLA method depends on the application scenarios, such as evaluation of cryptographic algorithms [8]–[10] and micro-architectural buffers [11].

A power side channel evaluation framework for symmetric key cryptography algorithms at the pre-silicon stage is proposed in [9], [10]. The initial step involves generating test patterns based on Hamming distance to introduce variations in power signatures. Then KL-divergence is used to evaluate the side channel leakage of the hardware implementation. Jayasena et al. [8] propose a framework for evaluating hardware implementations of asymmetric key cryptography algorithms. To preserve the timing information of the power traces during analysis, they propose a dynamic partition-based

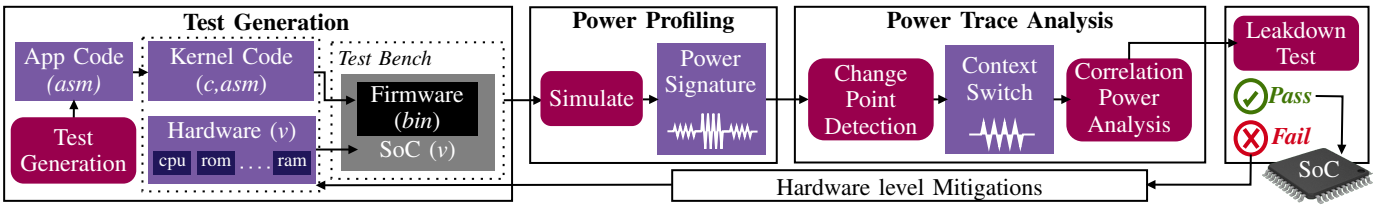


Fig. 4: Overview of information leakage assessment for security enclaves that consists of four major steps: (i) tests are generated to maximize the side-channel sensitivity, (ii) power model is generated by simulating the implementation, (iii) automated power analysis, and (iv) the divergence test is carried out to evaluate the side-channel leakage from the power signature.

differential power analysis technique using Welch's t-test. A software-based TVLA technique to evaluate branch prediction units is proposed in [11]. The authors have employed the Welch t-test to evaluate software cryptographic benchmarks.

Existing TVLA techniques cannot be directly applied to evaluate SoCs for the EvilCS vulnerability due to the fact that existing techniques focus exclusively either on hardware or on software. EvilCS spans across hardware-software boundary, requiring the simultaneous evaluation of hardware and software. For example, it involves both kernel code and application code to form the firmware during the evaluation process and test patterns need to be encoded into the application code. Similarly, the firmware needs to be compiled and programmed onto the hardware for evaluation. Therefore, the evaluation techniques need to be specifically tailored for the EvilCS vulnerability, as described in the next section.

III. Information Leakage Assessment

We propose a leakage assessment framework to assess the side-channel leakage of security enclaves. Figure 4 illustrates the main steps involved in the leakages assessment. First, we generate test cases to maximize side-channel sensitivity. Next, we construct the power signature of the implementation through simulation-based power profiling. We perform power trace analysis to identify the context switch interval. Next, we perform correlation power analysis to assess the information leakage. Finally, we perform a leakdown test to identify if the implementation has EvilCS vulnerability.

A. Test Generation for Side-Channel Sensitivity Maximization

The main objective of the information leakage assessment is to generate input test vectors to maximize the side-channel sensitivity of the underlying hardware. In case of EvilCS, input test cases are application programs. These programs should maximize the side-channel sensitivity of the system during a context switch call from the enclave kernel. In order to achieve this objective, we first need to generate multiple applications to write values directly to the system registers while maximizing side-channel sensitivity. Then we need to isolate the context-switching logic from the kernel and combine it with each application code to compile the firmware.

Application Code Template: In order to generate multiple (n) application codes, we have created an assembly template that can directly write values to GPR as illustrated in Figure 5a. Since the attacker is interested in the GPR, we focus on generating different combinations of values to be directly written into the registers, so that between each application the

```
# Load values from genHW()
lui  x1, {v_1[31:12]}
addi x1, x1, {v_1[12:0]}
lui  x2, {v_2[31:12]}
addi x2, x2, {v_2[12:0]}
...
lui  x31, {v_31[31:12]}
addi x31, x31, {v_31[12:0]}
nop
j os_kernel # return to os
```

(a) App Template (w=32bit)

```
#include <stdint.h>
#include "riscv.h"
int os_kernel(void)
{
    /* Context switch
    logic of kernel
    goes here */
    while (1) {}
    return 0;
}
```

(b) Kernel Wrapper

Fig. 5: Application template and kernel wrapper used for encapsulating different context-switch logic of security kernels.

power side-channel signature corresponding to each GPR is different. For this purpose, we use a naive approach presented in Algorithm 1. Inputs to the algorithm are the number of test cases required n, minimum hamming weight b to be maintained across register values, and the register width w. The actual value of n is determined by Equation 1, where $\Phi^{-1}(1-\frac{\beta}{2})$ and $\Phi^{-1}(\frac{\beta}{2})$ stands for upper and lower tails of the power trace distribution, respectively, while d, σ^2 and α represent effect size, the standard deviation of the power trace, and required statistical significance. Algorithm 1 generates a random number between 0 and 2^w and if it is within the required minimum hamming weight, it is appended to an array to be used within the application code. For each register in GPR, Algorithm 1 will generate an array of register values. Then for each `{v_i}` in the template, we assign values from each set of outputs from Algorithm 1.

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

Algorithm 1 Register value generation for application code

Require: Number of Tests n, Min HW b, Register Width w
Ensure: Hamming Weight Register Values C

```
function genHW(n, b, w)
    C ← ∅
    while n > |C| do
        a ← rand(2^w)
        if a ∉ C & HW(a) ≥ b then ▷ Hamming Weight ≥ b.
            C ← C ∪ {a}
        end if
        if |C| ≥ n then ▷ Enough values found for sample size
            Return C
        end if
    end while
end function
```

Kernel Code Isolation: We strip down the security kernel to isolate the context-switching logic from the other functionalities and use it inside the wrapper code as illustrated in

Figure 5b. This makes `j os_kernel` line in Listing 5a to perform a jump to the kernel to perform a context switch. Once n number of *assembly* programs are generated, we combine each application code with the striped version of the kernel to obtain n firmware versions. Next, we compile the firmware with the relevant *gcc* compiler to obtain n complete bare-metal executables on the hardware. Finally, the executable is compiled with the RISC-V hardware that is in hardware description language (HDL) and wrapped with a testbench code to obtain the compiled simulator. This process is automated for all the n implementation instances.

B. Simulation-based Power Profiling

After obtaining n implementation instances, we need to obtain the power model for each instance separately. We simulate each instance individually while dumping the simulation trace as a Value Change Dump (VCD). Next, each of the VCDs are converted into a power profile of the particular instance. Typically, side-channel patterns associated with the power attributes of hardware designs exhibit correlations based on the following two models [12]:

Switching Activity Model (SAM): SAM relates to the switching of internal signals of the device. Transitions from $0 \rightarrow 1$ and from $1 \rightarrow 0$ are deemed to consume higher power and emit more electromagnetic radiation compared to transitions from $0 \rightarrow 0$ and from $1 \rightarrow 1$.

Hamming Weight Model (HWM): HWM establishes a connection between the count of signals holding values 0 or 1 at a given instance and the overall power consumption of the device at that instance.

We use both power models to evaluate the presence of EvilCS vulnerability in security enclaves. We examine the VCD file and build the power profile models by considering both SAM and HWM by iterating through the signal value transitions and signal values during each clock cycle.

C. Power Trace Analysis

After obtaining the power profiles from the simulation, they need to be analyzed for potential information leakage. First, we employ change point detection to identify the areas related to the context switch. Then on the isolated power trace, we perform correlation power analysis to statistically identify the recoverability of register data.

Change Point Detection (CPD): Change point detection estimates the probability density function of data at various points and looks for abrupt changes in the estimated density. These abrupt changes can indicate potential change points in the data where something significant might have occurred. Due to the bulk register read and write, context-switch produces a high-density fluctuation in the power signature as illustrated by color ■ in Figure 2.

Let's represent the power signature of the device as a time series distribution $\{x_0, x_1 \ldots, x_n\}$. For this experiment of CPD, we construct two hypotheses H_0 and H_1 as *there is no change point in the power profile* and *there exists a change point in the power profile* respectively. Let $\hat{\mu}_1 (= \frac{1}{t_1} \sum_{i=1}^{t_1} x_i)$ and $\hat{\mu}_2 (= \frac{1}{t_2} \sum_{i=t_1+1}^{n} x_i)$ be mean before and after the change point. Then cumulative sum statistic (S_k) is introduced as a means of detecting changes in the data distribution as illustrated in Equation 2.

$$S_k = \max(0, S_{k-1} + x_k - \hat{\mu}) \tag{2}$$

If H_0 is true (no change point), then the S_k statistic before the potential change point will not exhibit significant deviations from zero. Conversely, if H_1 is true (change point exists), then the S_k statistic after the change point will likely exceed zero, signifying a significant change in the data distribution. Then based on the desired statistical significance level and the nature of the power profile, a threshold T is selected. It is often set to control the probability of making a Type I error (incorrectly detecting a change when there isn't one). In order to compute T, we randomly pick five samples from the distribution, manually perform a simple power analysis, and take the average as the change point threshold T. Therefore, a change point will be detected at k when $S_k > T$. In other words, when the cumulative sum statistic exceeds the threshold, it indicates the presence of a change point in the power trace which corresponds to a context-switch call from the kernel. The same technique is used to identify the endpoint of the context switch. Employing this process, a complete power signature during the context switch can be isolated from the entire power signature of the implementation. We repeat this procedure for n instances of the implementation to isolate context switch power signature related to each instance.

Correlation Power Analysis (CPA): In order to preserve a good resolution in the evaluation process, CPA is performed for each register involved in the context switching process. So far, we have n power profiles that only contain context switch power signatures, and with CPA we perform statistical analysis to evaluate the possibility of EvilCS vulnerability. First, each of the isolated context switch power traces needs to be segmented into equal-length sub-traces by dividing it by the number of registers involved in the context switch. This is possible since register read and write is a constant time operation and sequential operation as illustrated in Figure 3. Then each of the sub-trace is analysed against the known register values computed by Algorithm 1.

Let's consider the register R_j. Assume that the n sub-traces relevant for R_j as $(p_0, p_1 \ldots, p_n)$ and a set of corresponding known register values generated by Algorithm 1 as $(v_0, v_1, \ldots, v_n)$. For the CPA experiment, let's construct the hypotheses H_0 as *there is no correlation between the power consumption against the register values being computed in the trusted application* and H_1 as *there is a correlation between the power consumption and the register values*. Let the statistical significance for the experiment be α.

$$\chi^2 = \sum_{i=1}^{n} \frac{(P_i - E_i)^2}{E_i} \tag{3} \qquad E_i = \frac{\gamma(W(v_i)) \times \nu(P_i)}{\Lambda} \tag{4}$$

$$df = (|\gamma| - 1).(|\nu| - 1) \tag{5} \qquad p\text{–}value = 1 - CDF(\chi^2, df) \tag{6}$$

Then we use the Chi-squared statistic (Equation 3) to de-

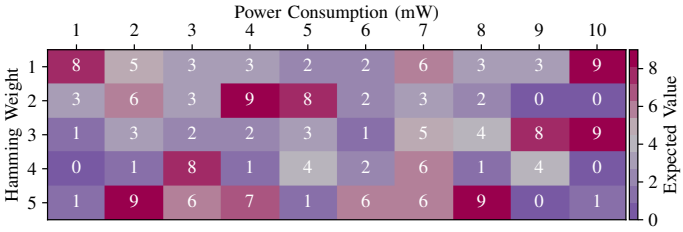


Fig. 7: Sample contingency table generated with power trace data (Pi) and hamming weight of register values ($W(v_i)$).

termine the correlation between the power value and the register value. In Equation 3, P_i corresponds to the peak power point ($P_i = max(\{p_i\})$) in the power trace corresponding to register R_j and expected power signature E_i is calculated from the contingency table constructed using the hamming-weight power model from v_i and p_i. Figure 7 illustrates an example contingency graph created from power traces p_i and hamming weights $W(v_i)$ of generated register values. The table is constructed by iterating through each $W(v_i)$ and filling with the peak power consumption values from corresponding power traces. Then E_i is calculated from Equation 4 where γ, ν, Λ represent row sum, column sum, and the total sum of the contingency table, respectively. This essentially translates hamming weight into an expected power consumption value. Next, we need to compute the degree of freedom df from Equation 5, where $|\gamma|$ and $|\nu|$ represent the number of rows and columns in the contingency table. For the example in Figure 7, $df = (5-1) \times (10-1) = 36$. Using the cumulative distribution function with the χ^2 Chi-Squared value and df degrees of freedom, we can determine the $p{-}value$ associated with the power trace from Equation 6. If $p{-}value \leq \alpha$, we reject the null hypothesis (H_0), which indicates a significant correlation between the power consumption and register value. If $p{-}value > \alpha$, we fail to reject the null hypothesis (H_0).

D. Classification using Leakdown Test

At this stage, we have established a relationship for each test instance with a statistical correlation between the power traces and the register values of the individual register. In the final step, we determine whether a register value could be potentially leaked from the power traces based on the CPA results of all the test instances. Note that based on the $p{-}value$, it can be concluded that either to reject H_0 or fail to reject H_0. Therefore, our classification method needs to classify an implementation based on whether it will leak information from a register or not. For this, we formulated the leakdown test, which will go through the results of each register level CPA and perform a family-wise rejection decision if at least one occurrence of H_0 rejection is found. This indicates that the implementation that rejects the null hypothesis for a particular register at least one time will fail the leakdown test and needs hardware mitigations to reduce the information leakage through power consumption. After applying the mitigations, the implementation needs to go through the assessment process from beginning to end until it can completely pass the leakdown test for all of its registers.

IV. Experiments

In order to perform information leakage assessment on real-world implementations, we have selected four security enclave kernels of Keystone [3], OpenMZ [5] (open source implementation of MultiZone [4] security kernel), Komodo^{rv} [13][1] and CertiKOS^{rv} [13][1] that are implemented for RISC-V instruction set architecture. We have isolated the context-switching logic from the kernel for evaluation purposes. For compiling the firmware we have used *riscv-gnu-toolchain*. Next, we have obtained four RISC-V SoC IP core implementations of PicoSoC, UervSoC, IObSoC, and VeeRwolf as target hardware designs. We used *Synopsys Design_Compiler* with *SAED90nm* CMOS technology for the synthesis of the design. We simulated hardware designs using *Synopsys VCS* to obtain the VCD signal dumps. Power signature construction is performed from *Synopsys vcd2saif* utility. For tasks such as test generation and leakage assessment, we developed customized *Python* scripts with the necessary statistics libraries. All the experiments were carried out in a server environment with Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz processor and 64GiB Memory.

A. Evaluation Parameters (n, T, α, w, b)

Based on the literature [8]–[10], we have selected the statistical significance α of 0.05. Since all the selected SoC implementations are 32 bits, we have selected the register with parameter (w) as 32. Column n in Table I presents the minimum number of traces required to evaluate different security enclave environments. Then the change point detection threshold T was also computed empirically from sample traces for each combination of implementations. Column T of Table I illustrates the computed T value for each implementation. The minimum hamming weight (b) was selected empirically as 5 from the sample power traces.

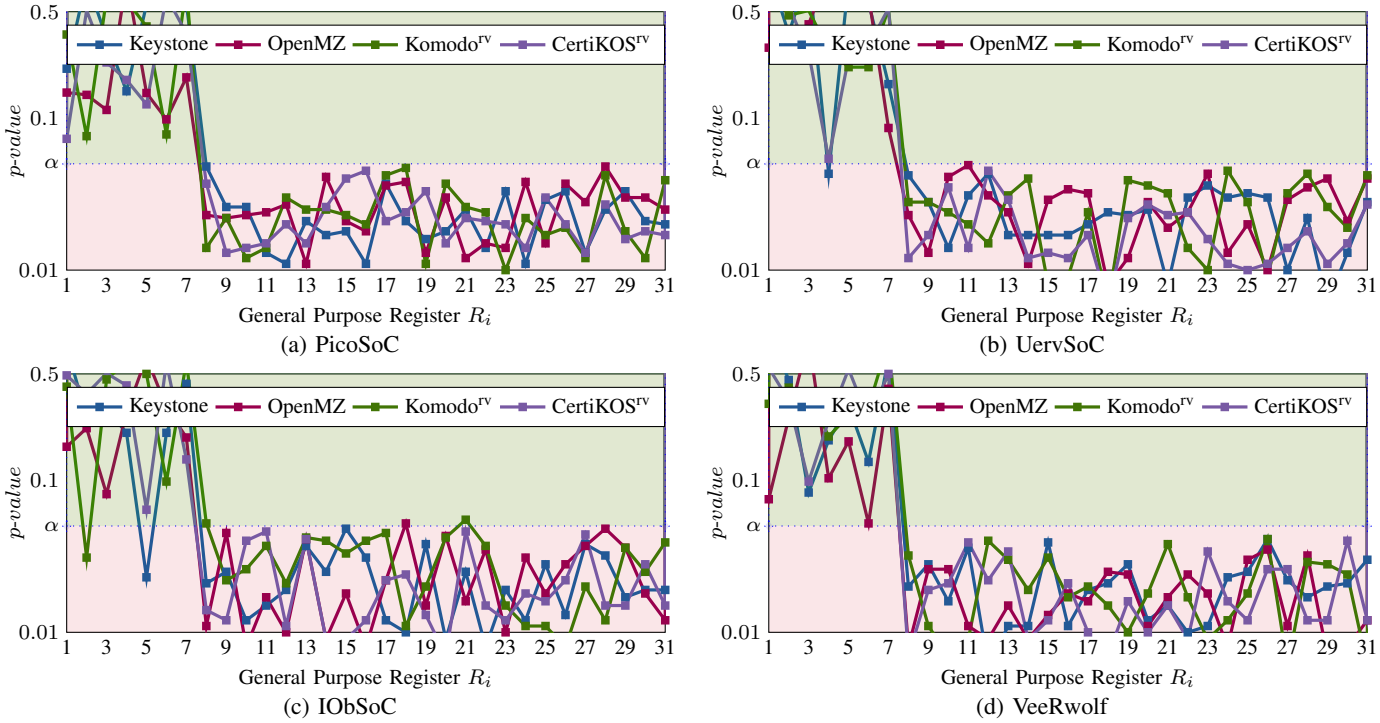
TABLE I: Sample size (n) and the change point detection threshold (T) for different combinations of security enclaves.

Hardware	**Context Switch Logic**							
	Keystone		OpenMZ		Komodo^{rv}		CertiKOS^{rv}	
	n	T	n	T	n	T	n	T
PicoSoC	6006	0.4	5691	0.3	5730	0.4	5829	0.4
UervSoC	6312	0.2	6198	0.2	6278	0.3	6910	0.2
IObSoC	4201	0.3	4448	0.3	4780	0.3	4583	0.3
VeeRwolf	7980	0.2	7495	0.2	8489	0.2	8358	0.2

B. Correlation Power Analysis on Registers

This section presents results about individual general purpose registers. Based on CPA results, we decide whether we can reject the null hypothesis H_0. Figure 8 presents the results for different configurations of security enclaves. Figure 8a illustrates the minimum $p{-}values$ observed during the experiments for four possible implementations of security enclaves with Keystone, OpenMZ, Komodo^{rv} and CertiKOS^{rv} kernels with the PicoSoC hardware. The same experiment was carried out through other security enclave configurations.

[1]Komodo^{rv} and CertiKOS^{rv} are retrofitted RISC-V versions of original Komodo and CertiKOS kernels [13]



(a) PicoSoC (b) UervSoC

(c) IObSoC (d) VeeRwolf

Fig. 8: Minimum $p\text{-}value$ observed for individual registers in different security enclave configurations (different kernels and hardware combinations). Registers under the line α fail the leakdown test and need mitigations to prevent information leakage.

Figure 8b, Figure 8c, and Figure 8d illustrate the results for UervSoC, IObSoC, and VeeRwolf, respectively. Any $p\text{-}value$ that is less than the selected significance value $\alpha = 0.05$, fails the leakdown test for the particular register. As demonstrated by results, general purpose registers ranging from $x8$ to $x31$ consistently fail the leakdown test showing a significant correlation between the power signature and occupied register value while most of the registers from $x1$ to $x7$, statistically fail to provide a conclusion. Note that in RISC-V architecture, $x0$ is a fixed zero value register and therefore we exempted it from evaluations. We observed that the reason for the weak correlation results for registers $x1$-$x7$ is that they are pointer registers. Unlike data registers of $x8$-$x31$ that store temporary data, operands, and results of calculations, pointer registers store information such as return address, stack pointer, etc. Although we directly write values in the application during the evaluation, later they get updated which affects the correlation analysis resulting values such that $p\text{-}value > 0.05$.

V. Conclusion

In this paper, we introduced EvilCS vulnerability to show that the power consumption of a security enclave can reveal sensitive information from trusted applications. Specifically, we demonstrate that the distinct consecutive memory reads and writes involved during context switching can leak register data as a power side-channel signature. We proposed an information leakage analysis framework to evaluate implementations that consist of different hardware and firmware configurations. Evaluation of sixteen combinations of RISC-V security enclaves reveals that a vast majority of general-purpose registers leak their values as side-channel signatures. This analysis is vital for designing secure and trustworthy systems. We briefly outline two potential countermeasures for RISC-V based security kernels. A designer can apply a firmware patch that needs modifications to the security kernel as well as the application code. Security kernel should temporarily disable system interrupts by manipulating the interrupt-enable bits in the Machine Status Register (`mstatus`). Once the hardware thread switches to the execution of the trusted application, it should perform its secret computation and once it is finished, the application should actively return to the kernel to re-enable the timer interrupts. An ideal fix against EvilCS would be to apply register masking as well as common blinding techniques for each general purpose register during pre-silicon design.

References

[1] Yangdi Lyu and Prabhat Mishra. A survey of side-channel attacks on caches and countermeasures. *HASS*, 2:33–50, 2018.

[2] Mahya Morid Ahmadi et al. Side-channel attacks on risc-v processors: Current progress, challenges, and opportunities. *arXiv*, 2021.

[3] Dayeol Lee et al. Keystone: An open framework for architecting trusted execution environments. In *EuroSys '20*, pages 1–16, 2020.

[4] Hex Five. Multizone security for risc-v, 2020.

[5] Henrik Karlsson. OpenMZ: a C implementation of the MultiZone, 2020.

[6] François-Xavier Standaert. Introduction to side-channel attacks. *Secure integrated circuits and systems*, pages 27–42, 2010.

[7] Aruna Jayasena and Prabhat Mishra. Directed test generation for hardware validation: A survey. *ACM Computing Surveys*, 2023.

[8] Aruna Jayasena et al. Test Vector Leakage Assessment on Hardware Implementation of Asymmetric Cryptography Algorithms. *TVLSI*, 2023.

[9] Nitin Pundir et al. Power side-channel leakage assessment framework at register-transfer level. *IEEE TVLSI Systems*, 2022.

[10] Tao Zhang et al. PSC-TG: RTL power side-channel leakage assessment with test pattern generation. In *ACM/IEEE DAC*, pages 709–714, 2021.

[11] Sarani Bhattacharya et al. Online detection and reactive countermeasure for leakage from bpu using tvla. In *VLSID*, pages 155–160. IEEE, 2018.

[12] Eric Brier et al. Correlation power analysis with a leakage model. In *CHES 2004*, pages 16–29. Springer, 2004.

[13] Luke Nelson et al. Scaling symbolic evaluation for automated verification of systems code with serval. In *SOSP*, pages 225–242, 2019.