Creating a Labeled Dataset for Fitting Deep Learning Model on The PicoRV RISC-V Core

Internship report by

Kamelia Zaman Moon

Supervised by

Ileana Buhan

In Partial Fulfillment of the Requirements for the

Degree of

Erasmus Mundus Joint Masters in IoT Cybersecurity

UNIVERSITÉ DE BRETAGNE-SUD

RADBOUD UNIVERSITEIT

June, 2023

ABSTRACT

As side-channel attacks (SCAs) continue to threaten cryptographic implementations on resource-constrained embedded devices, understanding how physical leakage emerges from microarchitectural behaviors is increasingly vital. This work presents a detailed evaluation of instruction-level leakage in masked cryptographic operations implemented on the lightweight PicoRV32 RISC-V core, as part of a broader effort titled "Creating a Labeled Dataset for Fitting Deep Learning Models on the PicoRV RISC-V Core." We investigate the practical leakage characteristics of several key gadgets: the ISW-masked AND, Toffoli-based reversible gates, and locality refresh operations, each implemented in multiple assembly-level variants. Using simulation and hardware traces, we perform correlation-based side-channel analysis (CSCA) under both Hamming Weight (HW) and Hamming Distance (HD) leakage models.

Our results reveal that HD leakage dominates, highlighting the importance of data transitions and register switching activity. While functionally equivalent, different implementation variants exhibit widely varying leakage profiles—underscoring the impact of instruction reordering, register usage, and operand dependencies. Notably, variants designed with more deliberate operand spacing and reduced reuse demonstrate significantly lower correlation, reinforcing the critical role of microarchitectural factors in side-channel resistance. These findings form the empirical foundation for the next phase of our project, which aims to construct deep learning models that can predict and generalize leakage across unseen code patterns and hardware conditions.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Number *Page*

Chapter 1

INTRODUCTION

As modern computing systems become increasingly pervasive in security-sensitive domains—from embedded IoT devices to secure communications—ensuring their resilience against side-channel attacks (SCAs) has become a critical concern. Side-channel attacks exploit physical leakages such as power consumption or electromagnetic radiation to infer secret data processed by a device, often without requiring access to the software stack or communication interfaces. These attacks are particularly threatening in constrained, low-power devices where traditional countermeasures like hardware isolation or robust cryptographic modules may not be feasible.

The RISC-V instruction set architecture (ISA), with its open-source model, modular design, and growing ecosystem, has emerged as a promising alternative to proprietary ISAs in both research and industry. However, its openness also means that potential vulnerabilities, including side-channel susceptibilities, are easier to study and exploit—posing a challenge and an opportunity for the security community. In lightweight RISC-V cores such as PicoRV32, implementing effective side-channel countermeasures requires a careful balance between performance, area, and security.

This report explores the side-channel resistance of masked cryptographic primitives implemented on RISC-V platforms, with a particular focus on the ISW masked AND operation, locality refresh operation, and its generalization to reversible computing gadgets like the Toffoli gate. The study combines simulation-based leakage modeling using tools like Qiling and PoMMES, with hardware-level trace acquisition via the CW-Husky and Saidoyoki platforms.

A key aim of this work is to assess the effectiveness of masking schemes at the instruction level, identify leakage sources introduced by microarchitectural behaviors, and explore equivalence-preserving variations in assembly-level implementations that might influence leakage profiles. By generating, executing, and analyzing thousands of input/output pairs, this study contributes to a deeper understanding of how theoretical countermeasures behave under practical constraints.

The rest of this report is organized as follows: the Background section outlines foundational concepts; the Literature Review contains outcomes of previous studies;

the Methodology describes the setup for simulation and measurement; Results and Analysis presents leakage findings and comparative evaluations; and the final sections discuss challenges, potential mitigations, and directions for future work.

Keywords

Side-channel analysis, correlation coefficient, power traces, Hamming Weight, Hamming Distance, leakage detection, masking countermeasures, RISC-V, secure implementation, CW-Husky, Saidoyoki, ChipWhisperer, Qiling, Toffoli gate.

Chapter 2

BACKGROUND

The increasing adoption of the RISC-V instruction set architecture (ISA) in embedded and Internet-of-Things (IoT) devices has raised significant concerns about side-channel security. While RISC-V offers openness, extensibility, and a lightweight footprint ideal for security-sensitive applications, its modularity and configurability also introduce unique challenges for implementing robust side-channel countermeasures.

Side-channel attacks (SCAs) exploit physical phenomena—such as power consumption, electromagnetic emissions, or timing variations—that occur during cryptographic computations. These attacks can reveal secret information (e.g., encryption keys) without exploiting software bugs. Among the most prominent are power analysis attacks, which include:

- Simple Power Analysis (SPA): Directly interprets patterns in the power signal.
- Differential Power Analysis (DPA): Applies statistical methods to large sets of traces to isolate key-dependent variations.

SCAs are particularly potent in lightweight microarchitectures where high integration and resource constraints often preclude robust hardware-based defenses.

A widely studied technique to mitigate SCAs is masking, which breaks the dependency between secret data and observable physical leakage by splitting sensitive variables into multiple shares. The ISW (Ishai-Sahai-Wagner) scheme is a prominent example of masking, where nonlinear operations like AND, locality refresh are carefully structured to maintain statistical independence of intermediate values.

Software masking on scalar processors such as RISC-V requires precise control over register usage and instruction flow to avoid unintended leakages due to glitches, transition activity, and compiler optimizations. This makes it necessary to analyze masked gadgets, such as the ISW masked AND, locality refresh, and Toffoli gates, at the instruction level.

RISC-V, being an open ISA, allows full access to hardware designs, enabling both pre-silicon and post-silicon leakage evaluation. Lightweight implementations such as PicoRV32 offer a realistic target for analyzing side-channel behavior in constrained environments. Pre-silicon evaluation often involves simulation tools or formal methods, while post-silicon analysis captures real physical traces from test-boards using devices like the CW-Husky.

Several frameworks have been developed for side-channel analysis of RISC-V and other ISAs:

- Saidoyoki: A platform designed for both pre-silicon and post-silicon leakage evaluation. It features two SoCs, one with a RISC-V core, and integrates trace capture and analysis pipelines.
- ChipWhisperer: A well-known platform for capturing power traces and performing side-channel attacks. The CW-Husky board is particularly suited for high-speed synchronized trace acquisition.
- Qiling: A dynamic binary instrumentation framework capable of simulating RISC-V binaries and logging register-level traces for side-channel behavior studies in software.
- PoMMES: A simulation-based framework for evaluating side-channel leakage via compositional power modeling, useful in pre-silicon design validation.

These platforms and tools enable in-depth assessment of cryptographic implementations by providing insights into how hardware structures, instruction scheduling, and masking schemes interact with physical leakage.

Chapter 3

LITERATURE REVIEW

Side-channel leakage assessment and mitigation in RISC-V architectures have been extensively studied due to their growing application in security-critical environments.

Gaspoz et al. [1] explored the application of threshold implementations (TI) to scalar micro-architectures to mitigate side-channel leakage in software implementations. Their study focused on adapting TI principles to scalar processors and addressing leakage effects using a two-share, randomness-free PRESENT cipher and Keccak-f. Implementations on RISC-V and ARM Cortex-M4 cores were evaluated using the glitch and transition extended probing model. The results indicated challenges in achieving side-channel resistance through software masking, particularly in scalar micro-architectures.

Kiaei et al. [2] presented the Saidoyoki platform, designed to evaluate side-channel leakage in both pre-silicon and post-silicon settings. The platform facilitates rapid leakage estimation from high-level design descriptions and supports detailed measurements on hardware prototypes. It hosts two different System-on-Chip (SoC) implementations, one based on a 32-bit RISC-V processor and another on a SPARC V8 processor, enabling a comparative analysis of leakage. The study emphasized the importance of early-stage leakage assessment, demonstrating that pre-silicon evaluations can provide enhanced test resolution and support root cause analysis, guiding the implementation of more effective mitigation strategies.

Miao et al. [3] developed RTL-PSC, an automated framework for assessing power side-channel leakage at the RTL level. The framework uses functional simulation to estimate power profiles and applies evaluation metrics such as Kullback-Leibler divergence and success rate to determine the vulnerability of hardware cryptographic designs. This automated approach allows for early detection of leakage issues, thereby enabling designers to apply countermeasures proactively.

Srivastava et al. [4] introduced SCAR, a pre-silicon power side-channel analysis framework that leverages graph neural networks (GNNs). SCAR converts RTL designs into control-data flow graphs, detecting modules prone to leakage. The

framework also features a deep learning-based explainer to interpret detection decisions, offering designers a clearer understanding of the vulnerabilities.

Despite significant advancements, certain gaps remain in current research. One major challenge is the integration of software and hardware countermeasures to create more comprehensive defenses. Additionally, the diversity of assessment methodologies and metrics complicates the comparison of results, indicating a need for standardized evaluation practices. Furthermore, as systems become increasingly complex, especially with multicore architectures, existing analysis frameworks must be scaled to accommodate larger and more heterogeneous systems. Lastly, the development of real-time monitoring and adaptive mitigation mechanisms remains limited, highlighting an area for future exploration.

Addressing these gaps could significantly enhance the resilience of RISC-V systems against side-channel vulnerabilities, particularly when employing lightweight and modular architectures like PicoRV32.

Chapter 4

METHODOLOGY

The methodology employed in this study integrates both pre-silicon simulation and post-silicon measurement techniques to assess side-channel leakage in RISC-V architectures. Specifically, we examine the leakage behavior of software-masked logical operations, such as ISW-masked AND, locality refresh, and generalized Toffoli gate gadgets, on a lightweight 32-bit RISC-V core. This dual-pronged approach ensures that both algorithmic and microarchitectural leakage vectors are systematically explored.

4.1 Target Platform and Gadget Design

All experiments target a 32-bit RISC-V SoC based on the PicoRV32 architecture, chosen for its lightweight, modular design and open-source accessibility. The gadgets are implemented in RISC-V assembly with minimal C wrappers and tested in both simulation (Qiling) and hardware 4.1 (CW-Husky + Saidoyoki board).

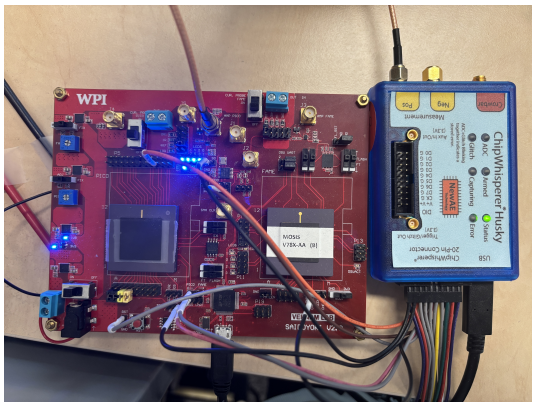

Figure 4.1: Saidoyoki board + CW-Husky

Three main gadgets were investigated.

ISW-masked AND Gadget

This gadget is based on the classical Ishai-Sahai-Wagner (ISW) masking scheme, which splits sensitive variables into two shares and uses a random mask to perform the AND operation securely. The gadget takes (a_0, a_1) and (b_0, b_1) as input shares

along with a random r, and outputs (c_0, c_1):

$$c_0 = (a_0 \cdot b_0) \oplus r \tag{4.1}$$

$$c_1 = (a_1 \cdot b_1) \oplus r \tag{4.2}$$

```
.section .text
.global isw_and
isw_and:
    # Inputs:
    # a0 = A0, a1 = A1
    # a2 = B0, a3 = B1
    # a4 = r

    mv t0, a0     # A0
    mv t1, a1     # A1
    mv t2, a2     # B0
    mv t3, a3     # B1
    mv t4, a4     # r

    # Compute partial products
    and t5, t0, t2      # t5 = A0 & B0
    and t6, t1, t3      # t6 = A1 & B1

    # Cross terms
    and t0, t0, t3      # t0 = A0 & B1 (reuse t0)
    and t1, t1, t2      # t1 = A1 & B0 (reuse t1)

    xor t0, t0, t1      # t0 = A0&B1 ^ A1&B0
    xor t0, t0, t4      # t0 = t0 ^ r

    xor t5, t5, t4      # C0 = A0&B0 ^ r
    xor t6, t6, t0      # C1 = A1&B1 ^ cross ^ r

    mv a0, t5           # Return C0
    mv a1, t6           # Return C1
```

```
    ret
```

Two variants of this gadget were developed by using temporary storage in memory and without masking:

- Version v0 – Register-Based ISW AND (Baseline): This version was implemented purely with register-level operations, where all intermediate computations were stored and propagated via general-purpose registers. It represents a clean baseline implementation adhering to ISW masking without any memory accesses or redundancy. This version minimizes memory-related noise and focuses solely on the instruction-level leakage due to register manipulation.
- Version v1 – Memory-Assisted ISW AND: In this version, temporary intermediate values were stored in memory (e.g., via sw and lw instructions) during computation. Although functionally equivalent to the baseline, the use of memory access introduced additional data movement and potentially new leakage vectors from memory bus activity, and load/store timing variations. This version was designed to reflect real-world coding patterns where intermediate values cannot always be retained in registers due to register pressure or modularity requirements.
- Version v2 – Unmasked AND: As a reference point, an unmasked version of the AND operation was implemented. This version directly computed the bitwise AND of the two original values without applying the ISW masking scheme. It served as a leakage-prone baseline to compare against the masked implementations and to observe how masking affects the power or simulation profiles.

This gadget is particularly sensitive to side-channel leakage due to the AND operation's non-linearity and thus serves as a valuable testbed for leakage assessment.

Toffoli Gate Gadget

The Toffoli gate is a reversible three-bit gate widely used in quantum and classical reversible computing. Its masked variant was implemented in RISC-V using ISW-protected ANDs for internal computations.

```
.section .text
```

```
.global toffoli_gate
# Input: a0, b0, c0, a1, b1, c1 stored in registers x10 - x15
# Temp register: x16 (R6)

toffoli_gate:
    and x16, x10, x11   # a0b0 -> R6 (x18)
    xor x12, x12, x16   # c0 + a0b0 -> R2 (x12)
    slli x10, x10, 2    # Shift R0 (x10)
    slli x14, x14, 2    # Shift R4 (x14)
    and x16, x10, x14   # a0b1 -> R6 (x18)
    srli x10, x10, 2    # Shift back R0 (x10)
    srli x14, x14, 2    # Shift back R4 (x14)
    slli x12, x12, 2    # Shift R2 (x12)
    xor x12, x12, x16   # c0 + a0b0 + a0b1 -> R2 (x12)
    srli x12, x12, 2    # Shift back R2 (x12)
    and x16, x13, x14   # a1b1 -> R6 (x16)
    xor x15, x15, x16   # a1b1 + c1 -> R5 (x15)
    slli x13, x13, 2    # Shift R3 (x13)
    slli x11, x11, 2    # Shift R1 (x11)
    and x16, x13, x11   # a1b0 -> R6 (x16)
    srli x13, x13, 2    # Shift back R3 (x13)
    srli x11, x11, 2    # Shift back R1 (x11)
    slli x15, x15, 2    # Shift R5 (x15)
    xor x15, x15, x16   # c1 + a1b1 + a1b0 -> R5 (x15)
    srli x15, x15,      # Shift back R5 (x15)
    xor x16, x16, x16   # Clear R6 (x16)
    mv x10, x15

    ret
```

Seven variants of the gadget were developed by permuting instruction order, using different temporary registers, alternating shifting sequence, adding redundant operations, introducing dummy operations, deliberating data dependency introduction, and adding temporary registers to assess their effect on leakage profiles:

- Version v0 - Toffoli (Baseline): The baseline version of the Toffoli gate gadget

is implemented in RISC-V assembly and serves as the reference for all subsequent variations. This implementation computes the masked Toffoli function using a single temporary register (x16) and applies bit-shift operations to simulate signal transitions, increasing switching activity for realistic leakage modeling.

- Version v1 - Instruction Reordering: In this version, independent instructions were reordered to change the temporal execution flow without affecting data dependencies. This helps assess whether shifting the timing of certain computations (e.g., bit shifts and logical ANDs) alters leakage characteristics under HD/HW models.

- Version v2 - Different Temporary Registers: This version of the Toffoli gate retains the same functional structure as the baseline (v0) but introduces a key variation: the temporary computations are performed using a different register (x17) instead of x16. The goal of this variation is to explore whether changing the register used for intermediate results affects side-channel leakage, either due to microarchitectural differences or register-specific switching patterns.

- Version v3 - Alternative Shifting Sequence: This version of the Toffoli gate introduces a subtle but meaningful change compared to the baseline implementation: the order in which the shift operations are applied and undone is altered. While logically equivalent, these changes can affect the side-channel characteristics due to different bit switching activity patterns, especially relevant under the Hamming Distance (HD) leakage model.

- Version v4 - Redundant Operation Insertion: Redundant logic (e.g., clearing a register using $addi\ x0, x0, 0$) was introduced intentionally. These instructions do not alter output but serve as computational noise, potentially masking meaningful transitions and suppressing side-channel signals.

- Version v5 - No-Op Jumps and Fake Dependencies: This version intentionally introduces control flow irregularities and artificial instruction-level delays in the form of no-op jumps and repeated XORs. These transformations aim to explore the effect of timing and data flow distortion on side-channel leakage while maintaining the same logical functionality.

- Version v6 - Deliberate Data Dependencies: Artificial data dependencies were created between instructions—for instance, saving a temporary result, then

restoring it before use. This variant investigates whether chaining operations influences leakage via register reuse and data propagation.

- Version v7 - Redundant Temporary Register Clearing: This variation modifies the original Toffoli gadget by inserting redundant clearing operations for the temporary register (x16). Although these operations do not affect the final output, they influence the internal data flow and timing, which can alter the power consumption or electromagnetic emission profiles, potentially affecting side-channel leakage.

Each version preserves the same logical functionality but introduces different microarchitectural side-effects (e.g., pipeline stalls or register reuse), which may affect power leakage. These versions allow for a fine-grained analysis of leakage at the instruction level.

Locality Refresh Gadget

The locality refresh is a lightweight masking primitive that re-randomizes shares without changing the underlying secret. It is particularly useful as a pre-processing or noise equalization step in masked computations

```
.section .text
.global locality_refresh
locality_refresh:
    # Inputs:
    # a0 = A0 (first share)
    # a1 = A1 (second share)
    # a2 = r (random mask)

    mv t0, a0        # t0 = A0
    mv t1, a1        # t1 = A1
    mv t2, a2        # t2 = r

    xor t0, t0, t2   # t0 = A0 ^ r
    xor t1, t1, t2   # t1 = A1 ^ r

    mv a0, t0        # a0 = refreshed A0
    mv a1, t1        # a1 = refreshed A1
```

```
ret
```

By XOR-ing both shares with the same random mask r, the sum of the two shares remains constant:

$$A'_0 \oplus A'_1 = (A_0 \oplus r) \oplus (A_1 \oplus r) = A_0 \oplus A_1 \tag{4.3}$$

This gadget was assessed both as an independent operation and as part of larger masked circuits, especially to test its impact on uniformity and leakage suppression. Three variations of this gadget were made by reducing the number of operations, swapping registers, and mixing register uses:

- Version v0 - Locality Refresh (Baseline): This one has a simple and clean implementation, minimal control-flow complexity. Predictable register usage and minimal operation interleaving make this vulnerable to side-channel attacks, especially under Hamming Distance or Hamming Weight models, since the data flow remains structured and deterministic.
- Version v1 - Reduced Operations: This version eliminates intermediate register moves and performs masking directly on input registers (e.g., xor a0, a0, a2), thereby reducing the number of instructions and transitions. However, this can increase leakage on the input registers directly, since sensitive transitions (e.g., a0 $\rightarrow$ a0 ^r) are now visible at the architectural register level, without indirection.
- Version v2 - Swapped Register Usage: This version swaps the roles of temporary registers (t0, t1, t2) and input registers, assigning shares to different temporary registers or in a different order. Depending on pipeline structure and operand forwarding in the processor core (e.g., PicoRV32), this can lead to different temporal leakage patterns.
- Version v3 - Mixed Register Use: This introduces irregular use of registers. It refreshes one share (a1) in-place, while the other (a0) uses a temporary register (to). This creates asymmetry in the instruction sequence and register pressure.

4.2 Input Vector Generation

To simulate real-world cryptographic scenarios, 5000 input vectors were generated for each gadget using Python scripts. Each vector contains random 32-bit values for inputs and masks. For the ISW-masked AND, the CSV file contains fields such as $a_0, a_1, b_0, b_1, r, c_0, c_1$, and $expected\_and$. The Toffoli gate vector generation followed a similar structure with additional intermediate values such as $a_0, b_0, c_0, a_1, b_1, c_1, x_12, x_15$. The locality refresh vector generation also followed a similar structure with a_0, a_1, r, c_0, c_1, and $expected\_value$.

4.3 Pre-Silicon Simulation with Qiling

To analyze potential side-channel leakage before hardware implementation, Qiling, a lightweight and extensible dynamic binary emulation framework, was employed to simulate RISC-V binaries in a pre-silicon setting. This approach enabled detailed observation of the program execution without the need for physical measurements, providing early insights into how sensitive data propagates through registers during masked computations.

This plugin recorded the values of general-purpose registers after each instruction, capturing fine-grained changes across the entire execution flow. Registers such as x10 to x17, typically used for passing function arguments and holding intermediate values, were of particular interest, as they frequently carried sensitive data or masked shares.

Each instruction was also categorized according to its operation type—for example, logical operations like AND and XOR, or shift operations like SLLI and SRLI. By tagging each instruction in this way, it became possible to correlate certain instruction classes with higher or lower levels of data-dependent variation. This classification facilitated a more focused analysis of how different sequences of instructions might contribute to side-channel leakage.

During simulation, the plugin generated execution traces in a structured CSV format. The hook code for this is shown below:

```
def code_hook(ql: Qiling, address: int, size: int, md: Cs) -> None:
    """
    This function gets executed for every instruction.
    We're writing the PC, instruction, and all register
    contents into a file
```

```
        """
        row=[address]
        for reg in riscv_registers:
            row.append(hex(ql.arch.regs.read(riscv_registers[reg])))
        writer.writerow(row)


    def code_hook(ql: Qiling, address: int, size: int, md: Cs) -> None:
        """
        This function gets executed for every instruction.
        We're writing the PC, instruction, and all register
        contents into a file
        """
        buf = ql.mem.read(address, size)
        # Preparing the instruction definition
        read_ins=[]
        for insn in md.disasm(buf, address):
            read_ins.append(hex(insn.address)) #PC
            read_ins.append(buf.hex()) #machine code
            read_ins.append(insn.mnemonic) #mnemonic
            #read_ins.append(return_instruction_type(insn.mnemonic))
            read_ins.append(return_instruction_type(insn.mnemonic.upper())[0])
            read_ins.append(insn.op_str) #parameters
            #'Machine','Ins','Type','Operands'

        #print(read_ins)
        #ql.log.debug(f"====== PC: {ql.arch.regs.pc: #x} ======")

        row=read_ins #initialize the row with instruction information
        for reg in riscv_registers:
            row.append(hex(ql.arch.regs.read(riscv_registers[reg])))
        writer.writerow(row)
```

Each row corresponded to a single instruction execution and included the program counter, the executed instruction, and the values of key registers at that moment. These traces provided a high-resolution view of data flow within the processor for

each input set. To ensure consistency with post-silicon experiments, the same input vectors—both fixed and randomized—were used as in the hardware-based leakage measurements. This alignment made it easier to compare behavioral patterns across simulation and real execution.

Following the pre-silicon simulation using Qiling, the next step involved applying side-channel leakage models to the recorded execution traces in order to analyze the potential for data-dependent leakage. These models—Hamming Weight (HW) and Hamming Distance (HD)—are widely used in side-channel analysis to approximate how internal data transitions influence observable side-channel signals like power consumption.

The Hamming Weight model assumes that the power consumption of a register is correlated with the number of bits set to 1 in its binary value. In this work, the HW model was applied to the register values logged at each instruction step. For every trace, each relevant general-purpose register was analyzed by computing the Hamming Weight of its value. These HW values were appended as new columns in the CSV-formatted execution traces.

This modeling allowed the visualization and statistical analysis of how sensitive intermediate values—such as masked shares or partial computations—may influence power consumption under the assumption that leakage correlates with bit-level population count. Because the masking schemes (like ISW) aim to decorrelate power from sensitive data, HW modeling helped identify whether such schemes were effective in simulation.

The HD model, on the other hand, captures the number of bit transitions between successive values of a register and is often considered more accurate for dynamic power consumption, especially in real hardware. To apply this model, the register values at two consecutive time steps were compared, and the number of differing bits was computed.

Formally, for each register R, the Hamming Distance was computed as: $HD(R_t, R_{t-1}) = HW(R_t \oplus R_{t-1})$ where R_t and R_{t1} are the register values at time t and t1, respectively, and $\oplus$ denotes the bitwise XOR operation.

This model was particularly useful for understanding leakage caused by transitions in key registers—especially those holding sensitive masked values or being overwritten with recombined outputs. The HD traces provided a richer perspective on how masking might reduce leakage even in the presence of high switching activity.

```
selected_registers = ['zero', 'ra', 'sp', 'gp', 'tp', 't0', 't1',
                      't2', 's0', 's1', 'a0', 'a1', 'a2', 'a3', 'a4',
                      'a5', 'a6', 'a7', 's2','s3', 's4', 's5', 's6',
                      's7', 's8', 's9', 's10', 's11', 't3', 't4',
                      't5', 't6']


def HD(a,b):
    """
    Computes the hamming distance between two integers
    """
    hd = 0
    diff = a^b
    while diff:
        hd += diff & 1
        diff >>= 1
    return hd


def HW(a):
"""
Computes the hamming weight of an integer
"""
return sum([a&(1<<i)>0 for i in range(32)])
```

4.4 Post-Silicon Measurement with CW-Husky and Saidoyoki

For empirical validation, the gadgets were deployed onto a physical testbed comprising:

- CW-Husky: A modern side-channel acquisition platform from NewAE, capable of capturing high-resolution power traces.
- Saidoyoki SoC: An SoC developed for secure design evaluation, featuring a RISC-V softcore and integrated I/O for triggering and communication.

Each input vector is sent over UART to the target SoC, which computes the gadget logic and returns the result. The Husky simultaneously captures the power trace during the computation, synchronized using GPIO trigger signals. The system setup supports:

- Trace alignment using the rising edge of trigger signals.
- Consistent clock configuration (10 MHz internal for SoC)
- High sampling rate (200 MS/s)

The stimulus vectors and capture configuration are defined in JSON and CSV files, respectively, and used by ChipWhisperer's Python API.

Chapter 5

RESULTS AND ANALYSIS

This section presents the empirical observations derived from pre-silicon simulation traces and leakage modeling. The aim was to assess and compare the side-channel leakage behavior of multiple RISC-V assembly implementations of masked logic gadgets, focusing on ISW-based AND gates, Toffoli gate variants, and locality refresh operations.

5.1 Execution Trace Overview

Execution traces were collected using the Qiling framework for each gadget and variation. The Qiling-based simulation plugin successfully captured instruction-by-instruction register states during program execution. These states were exported to CSV files for each input vector, enabling fine-grained inspection of data flow and operations across registers.

Some sample traces are shown below:

PC	Ins	Operands	a0	a1	a2	a3	a4	a5	a6
0xc94	and	a6, a0, a1	0x473c08	0xa047ef4	0x1375d2	0xe0278fb	0x9c2ef69	0x32c2561	0x0
0xc98	xor	a2, a2, a6	0x473c08	0xa047ef4	0x1375d2	0xe0278fb	0x9c2ef69	0x32c2561	0x4084c
0xc9c	slli	a0, a0, 0x2	0x473c08	0xa047ef4	0x1375d2	0xe0278fb	0x9c2ef69	0x32c2561	0x4084c
0xca0	slli	a4, a4, 0x4	0x473c08	0xa047ef4	0x1375d2	0xe0278fb	0x9c2ef69	0x32c2561	0x4084c
0xca4	and	a0, a1, a2	0x1cf201f	0xa047ef4	0x1371da	0xe0278fb	0x738bda	0x32c2561	0x4084c

Table 5.1: Instruction Table For Toffoli

PC	Ins	Operands	a0	a1	a2	a3	a4
0xc94	addi	sp, sp, -0x10	0x44459e24	0x8a3587bd	0x2ee9fa44	0x82379a7a	0x37
0xc98	sw	a0, 0(sp)	0x44459e24	0x8a3587bd	0x2ee9fa44	0x82379a7a	0x37
0xc9c	sw	a1, 4(sp)	0x44459e24	0x8a3587bd	0x2ee9fa44	0x82379a7a	0x37
0xca0	sw	a2, 8(sp)	0x44459e24	0x8a3587bd	0x2ee9fa44	0x82379a7a	0x37
0xca4	sw	a3, 0xc(sp)	0x44459e24	0x8a3587bd	0x2ee9fa44	0x82379a7a	0x37

Table 5.2: Instruction Table For ISW AND

PC	Ins	Operands	t0	t1	t2	a0	a1	a2
0xc94	mv	t0, a0	0x0	0x0	0x0	0x1548398b	0x874231c4	0x5ee17ffb
0xc98	mv	t1, a1	0x1548398b	0x0	0x0	0x1548398b	0x874231c4	0x5ee17ffb
0xc9c	mv	t2, a2	0x1548398b	0x874231c4	0x0	0x1548398b	0x874231c4	0x5ee17ffb
0xca0	xor	t0, t0, t2	0x1548398b	0x874231c4	0x5ee17ffb	0x1548398b	0x874231c4	0x5ee17ffb
0xca4	xor	t1, t1, t2	0x4ba94670	0x874231c4	0x5ee17ffb	0x1548398b	0x874231c4	0x5ee17ffb

Table 5.3: Instruction Table For Locality Refresh

This pre-silicon data proved valuable for analyzing gadgets in isolation before conducting any physical measurements.

5.2 Leakage Modeling Results

To evaluate the presence of potential side-channel leakage, Hamming Weight (HW) and Hamming Distance (HD) models were applied to the simulated execution traces. These models allowed us to estimate how much information an attacker might infer from observing transitions on the datapath or register values during execution.

ISW AND Gadget

The ISW-masked AND gadget, implemented with two shares and a random mask, showed moderate variation in HW and HD values across executions.

Here are the plots, comparing HD (Hamming Distance) and HW (Hamming Weight) simulated traces for different versions (v0, v1, v2; where v0 is the original one and others are variations) of the ISW-masked AND gadget:

Figure 5.1 and 5.2 show the simulated Hamming Distance for ISW AND v0, v1, and v2, across instruction sequences. All versions exhibit noticeable variation in HD values across the instruction sequence. The HD values generally oscillate, indicating switching activity that depends on operand changes.

The original or ISW AND v0 shows clear alternating peaks and valleys. The pattern suggests that instruction-level changes affect multiple bits, leading to frequent switching. Some instructions trigger high HD (e.g., index 1 and 2), indicating critical transitions.

ISW AND v1 shows more frequent fluctuations with a burst of high HD values between instructions 6–10. It suggests that this version may have less stable data movement or more correlated intermediate values. Later part of the trace is relatively more stable.

ISW AND v2 shares a pattern similar to v0, but with slightly fewer peaks. The lower variation toward the end might suggest better masking uniformity or less switching.

Higher HD implies more bit-level switching, which can correlate with higher potential for leakage if the switching is data-dependent. Among these, v1 may be the most concerning from a leakage perspective due to its volatility.

Figure 5.3 and 5.4 show the simulated Hamming Weight for the same three versions. HW traces are smoother than HD traces, as HW reflects static values rather than

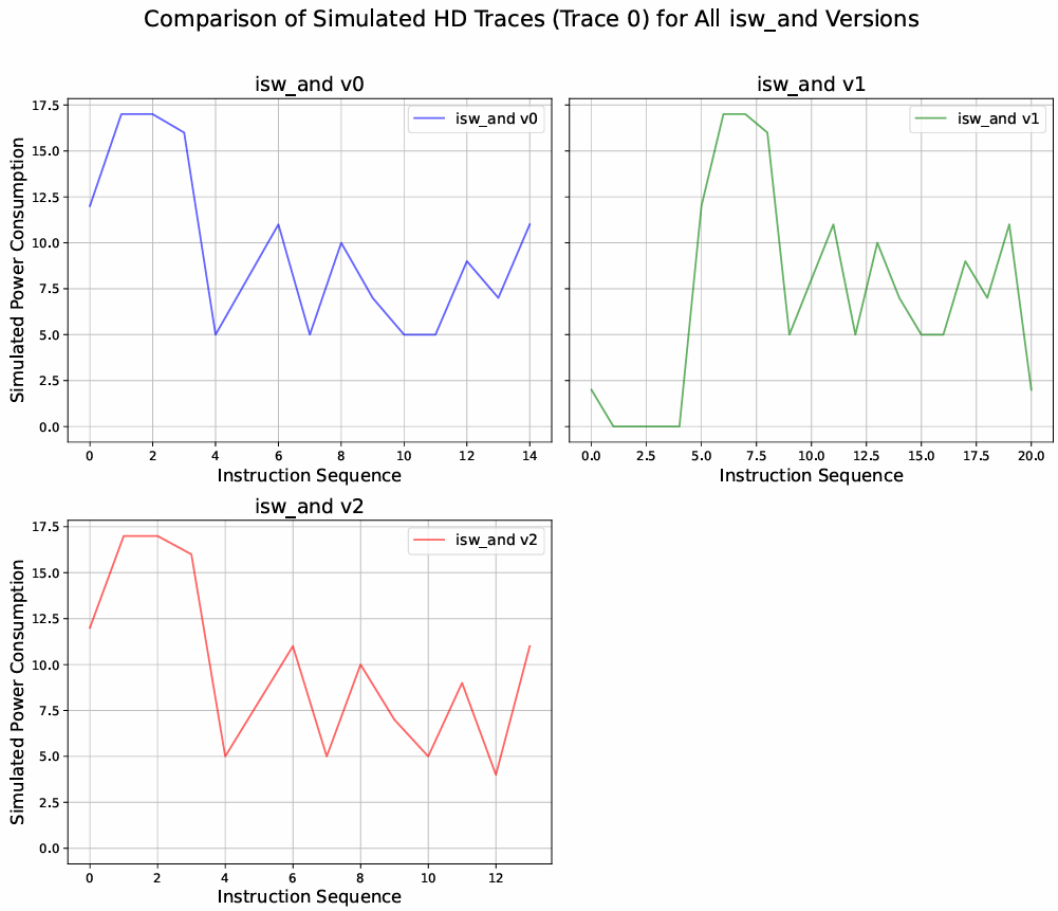

Figure 5.1: Simulated HD Traces for ISW AND

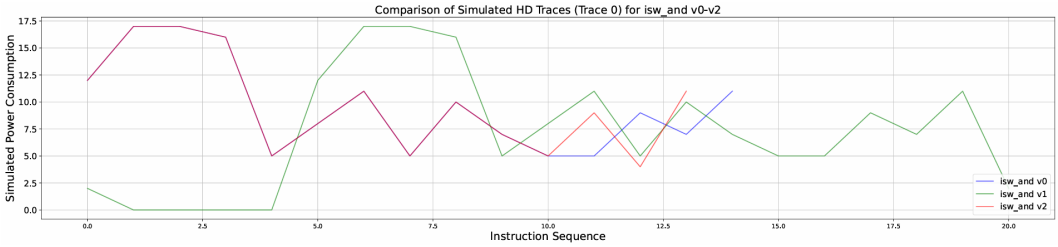

Figure 5.2: Simulated HD Traces in a Single Plot for ISW AND

transitions. All three versions show a rising pattern up to a certain instruction index, then plateau or slightly fall.

ISW AND v0 or the original version starts around 95 and peaks close to 185 around instruction 7. After peaking, the HW slightly dips and stabilizes, indicating less variation in bit-wise values.

ISW AND v1 displays a similar peak profile, but reaches a plateau later (around

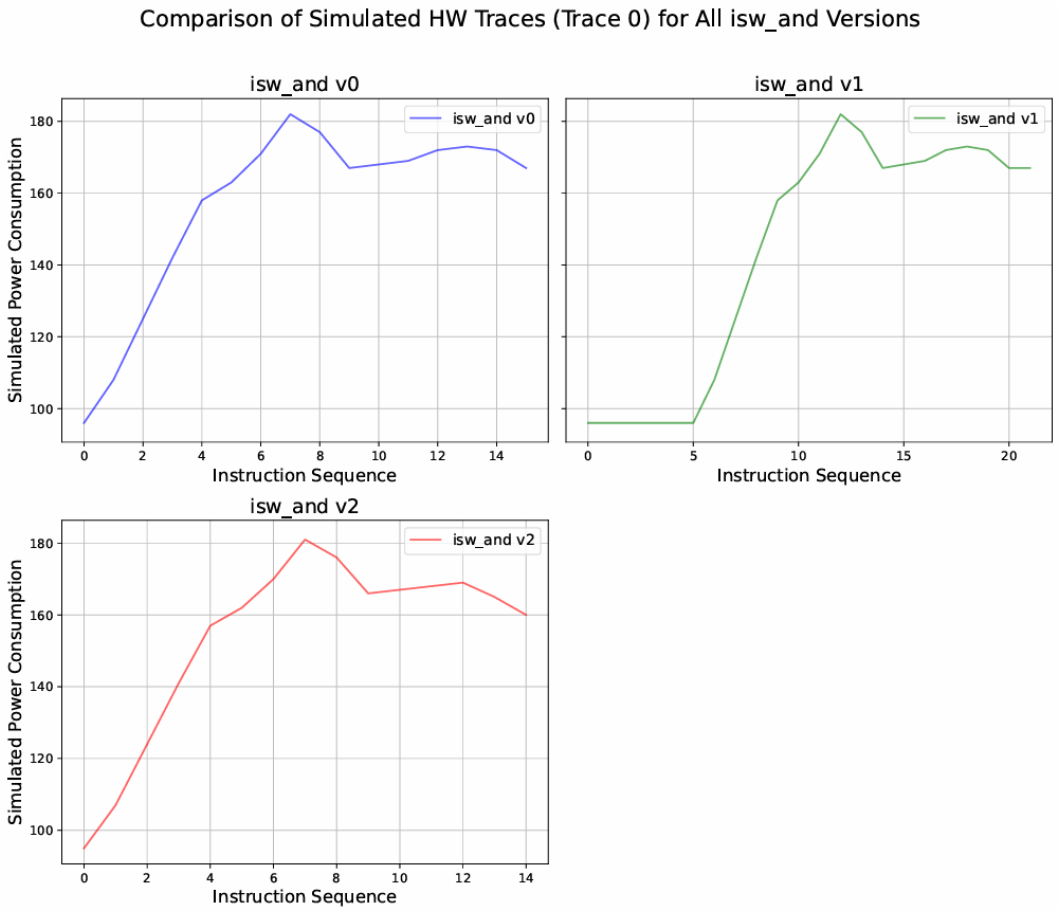

Figure 5.3: Simulated HW Traces for ISW AND

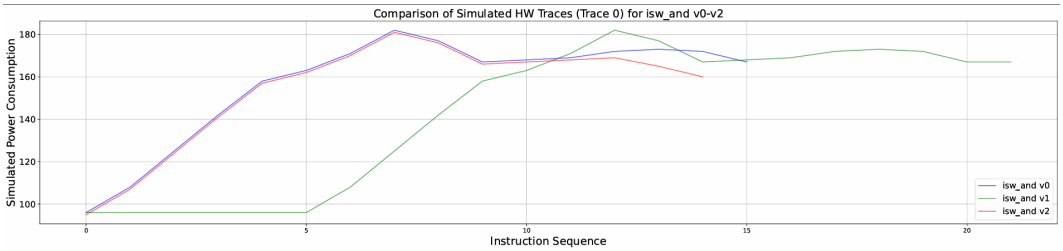

Figure 5.4: Simulated HW Traces in a Single Plot for ISW AND

instruction 11). Overall smoother, but the late rise indicates longer propagation of intermediate values.

ISW AND v2 is very similar to v0 in structure. It has a slightly delayed peak, but not significantly different in max HW.

Higher HW means more bits set to 1, which could be relevant in scenarios where the number of 1s correlates with power draw. v1 again stands out by being more delayed in stabilization, which might expose more exploitable behavior if an attacker observes prolonged execution.

Toffoli Gate Gadget

Here is a detailed analysis of the simulated Hamming Distance (HD) traces for each Toffoli gate version (v0–v7), as visualized in figure 5.5 and 5.6:

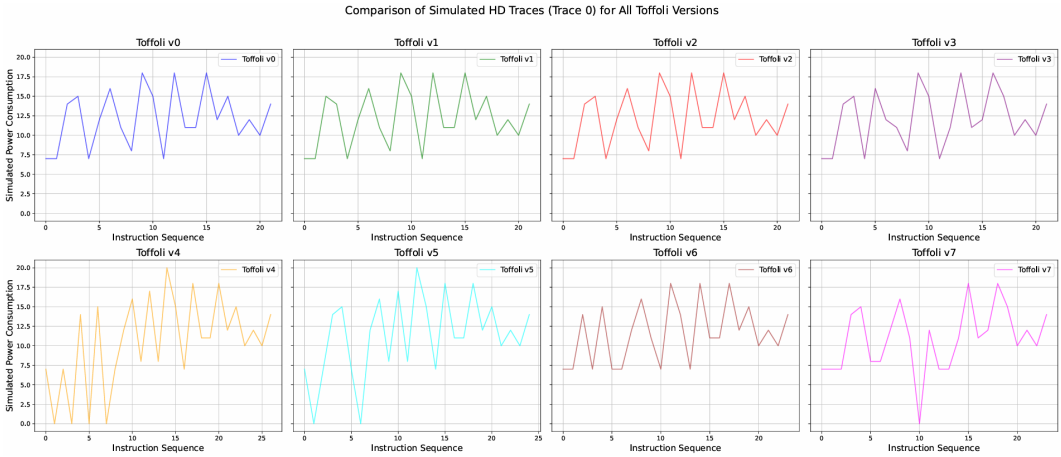

Figure 5.5: Simulated HD Traces for Toffoli Gate

Toffoli v0 is the original one that displays a moderate fluctuation pattern with values generally between 7 and 16. The trace exhibits consistent behavior with no extreme outliers, making it a baseline for leakage comparison. This suggests a regular

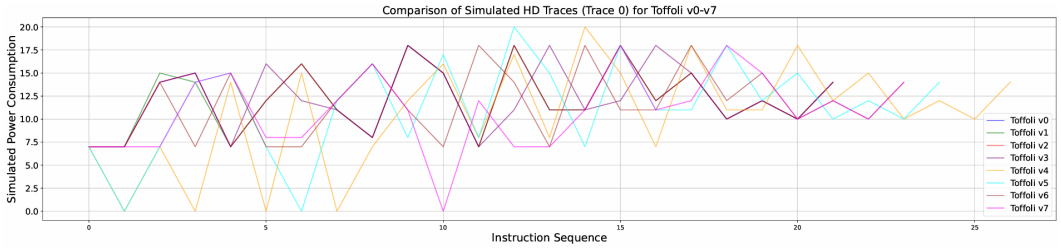

Figure 5.6: Simulated HD Traces in a Single Plot for Toffoli Gate

and predictable instruction-to-instruction transition profile, which might be more vulnerable to correlation-based attacks.

Toffoli v1 has similar magnitude range as v0 but with noticeably different sequence and shape. The instruction permutation appears to alter the data dependency graph, slightly affecting when high transitions happen.

Toffoli v2 is very similar to v1 and v0 in shape, suggesting that changing registers alone does not significantly alter the leakage unless it impacts register reuse and bit flips. This shows relatively regular behavior but lacks major noise or entropy, implying predictability remains.

Toffoli v3 has slight deviations in peak alignment compared to earlier versions. Its minor reordering leads to redistribution of switching activity, but overall profile remains consistent. This indicates that simple shifts in bitwise operation order offer strong leakage.

Toffoli v4 displays higher variance, with sharp peaks (e.g., above 18) and occasional dips (below 5). Its redundant instructions appear to increase randomization of switching behavior, potentially acting as a simple masking or obfuscation technique.

In toffoli v5, the trace is less smooth, with visible sharp drops and spikes across the sequence. Here, inserted jumps and dependencies result in irregular switching patterns, contributing to execution delays and misaligned transitions.

Toffoli v6 exhibits a pattern with frequent mid-level peaks and interspersed low-activity sections. Reusing and adding temporaries likely dilutes the bit transitions across multiple registers, increasing leakage per instruction.

Toffoli v7 is notable for flat zero or very low points at regular intervals—indicative of deliberate clearing of registers.

Figure 5.7 and 5.8 shows simulated Hamming Weight (HW) traces for Toffoli gate versions v0–v7.

Here is a detailed analysis that reveals how data-dependent leakage manifests across different instruction sequences and gadget variants:

Toffoli v0 shows a consistent range of values around 132–136 for most of the execution, with a sharp drop at the end (approximately 122). The sharp fall in the last few instructions may correspond to a data recombination or result write-back, where register content is reset or compressed, potentially revealing output

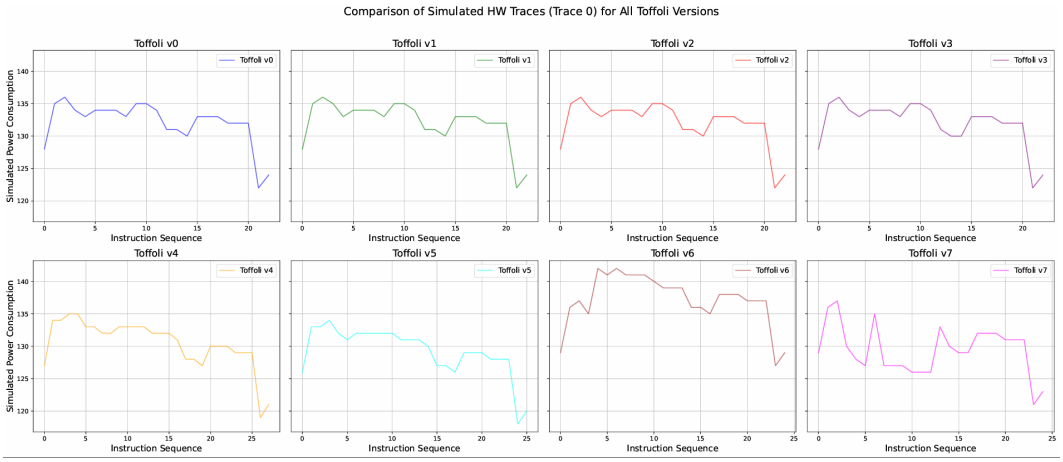

Figure 5.7: Simulated HW Traces for Toffoli Gate

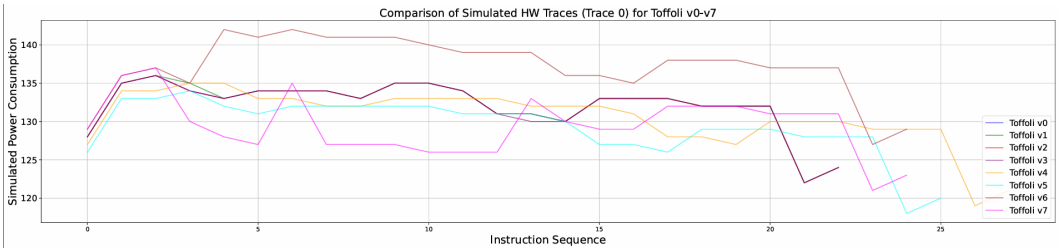

Figure 5.8: Simulated HW Traces in a Single Plot for Toffoli Gate

bit patterns. The rest of the trace appears stable, but the lack of noise and visible patterns suggest data-dependent activity is clearly observable.

Toffoli v1 is very similar overall profile to v0, indicating that mere instruction order changes don't significantly impact Hamming Weight leakage. Leakage remains highly structured and correlatable, indicating that sensitive intermediate values are not sufficiently masked or randomized.

In toffoli v2, the pattern is nearly indistinguishable from v1, confirming that changing temporary registers without masking doesn't impact leakage noticeably. All peaks and drops occur at similar instruction indices as in v0/v1, suggesting similar register transitions and switching activity.

In toffoli v3, again, very similar to v0-v2, indicating that variations in bitwise shifts do not change register-level Hamming Weight significantly. This suggests that the Hamming Weight model is not sensitive to operation permutation alone when overall bit transitions remain unchanged.

In toffoli v4, the trace shows more fluctuations across the instruction sequence,

especially in the middle and later regions. Here, redundant operations add noise, potentially masking the correlation between specific values and observed trace power, though leakage remains visible. The additional instructions cause longer execution (more data points), slightly diffusing signal alignment.

Toffoli v5 shows subtle differences compared to v4, with irregular spacing between peaks. Dummy operations seem to delay or desynchronize critical transitions, which can interfere with side-channel alignment techniques.

Toffoli v6, exhibits slightly higher initial activity (approximately 139) and more pronounced undulations compared to v0–v5. This may indicate additional register content transitions, increasing power fluctuation due to added temporary values. It shows a bit more spread, which could obfuscate specific value transitions if combined with masking.

Toffoli v7 shows the most irregular and noisy trace among all. Clear signs of zeroing operations are visible—drop in HW values at regular points, breaking the continuity of power. This version introduces artificial transitions to fixed values (e.g., 0), which can confuse Hamming Weight-based leakage models.

Locality Refresh Gadget

The locality refresh routine (a0 ^= r; a1 ^= r) showed a more uniform HW/HD profile. Because the transformation applies an identical XOR operation to both shares, its leakage is primarily dependent on the random mask r, which was fixed during some simulations.

Here is a detailed analysis of the simulated Hamming Distance (HD) traces for each Locality Refresh version (v0-v3), as shown in figure 5.9 and 5.10.

Locality Ref v0 is the original one where steep transitions are observed around instruction indices 1–2. Structured and regular use of mv and xor with temporary registers leads to predictable transitions.

Locality Ref v1 has a flat trace. Although this version has fewer instructions, it operates directly on input registers (a0, a1). This results in sensitive transitions being directly exposed, but that might not be captured well in this simulated HD model.

Locality Ref v2 is very similar to v0. The use of different registers for the same logical roles causes changes in timing and transition locations.

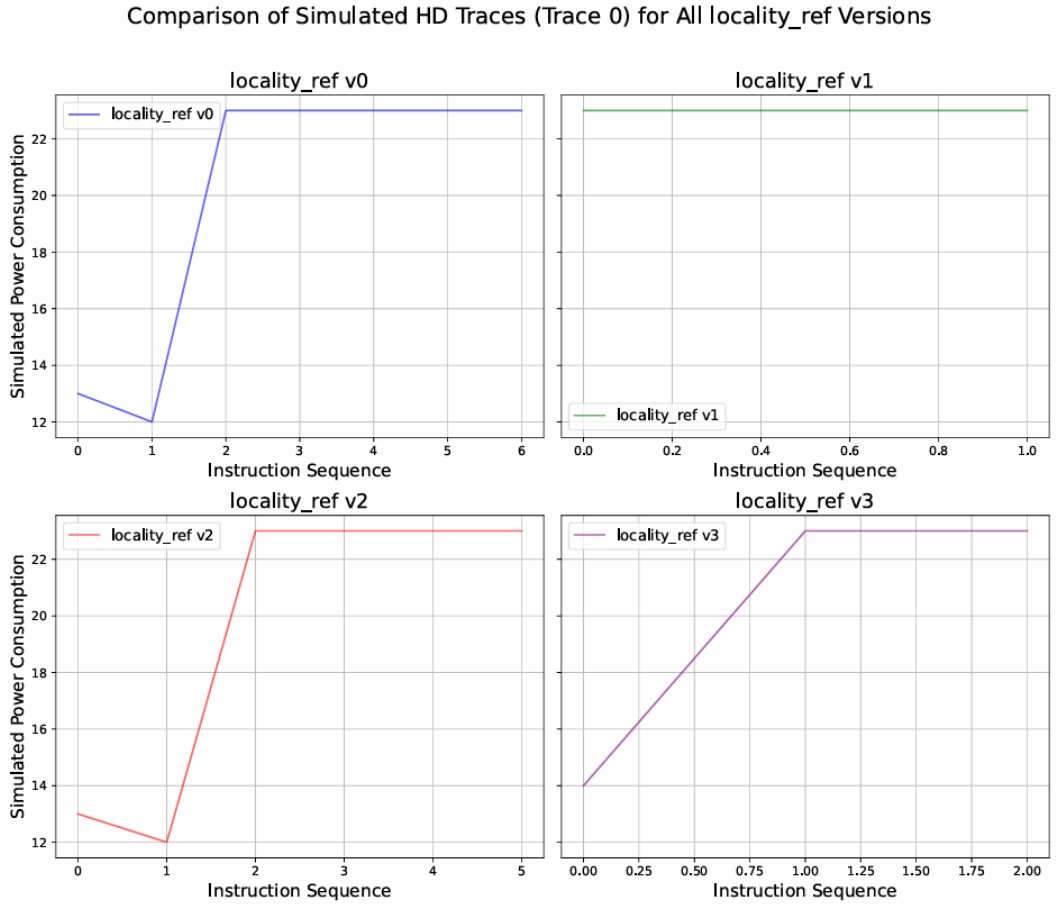

Figure 5.9: Simulated HD Traces for Locality Refresh

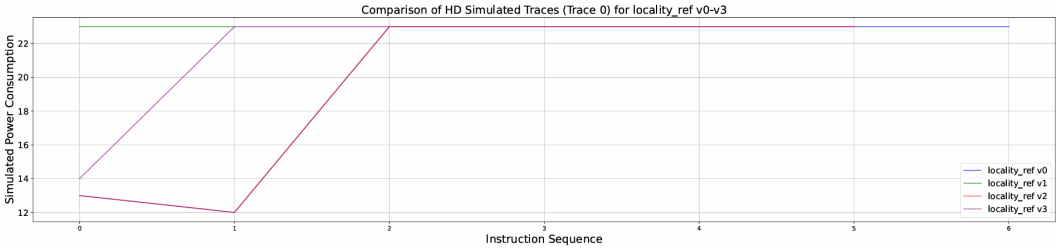

Figure 5.10: Simulated HD Traces in a Single Plot for Locality Refresh

Locality Ref v3 has a gradual increase over time. It has a more complex pattern than others. The asymmetric refreshing (one share in-place, one via temp register) causes irregular transitions.

Figure 5.11 and 5.12 shows simulated Hamming Distance (HD) for Locality Refresh v0-v3.

Locality Ref v0 or baseline version shows a clear and steady increase in power

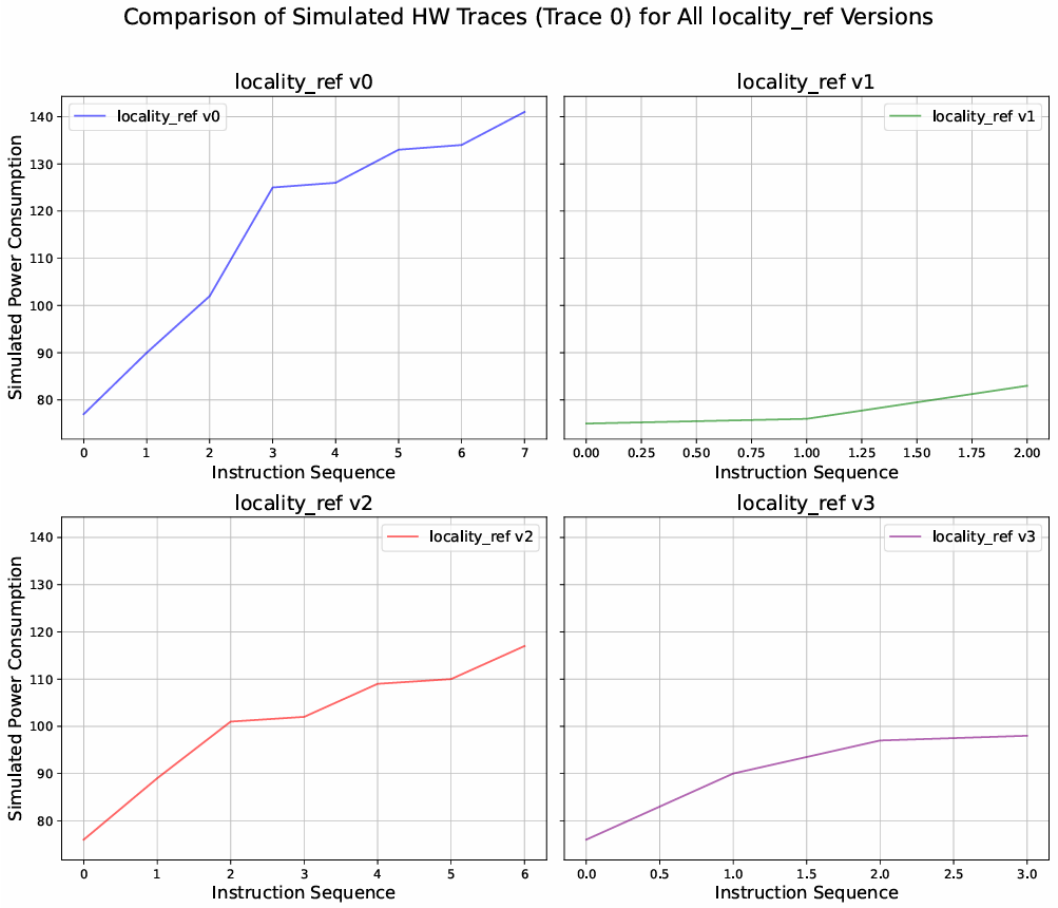

Figure 5.11: Simulated HW Traces for Locality Refresh

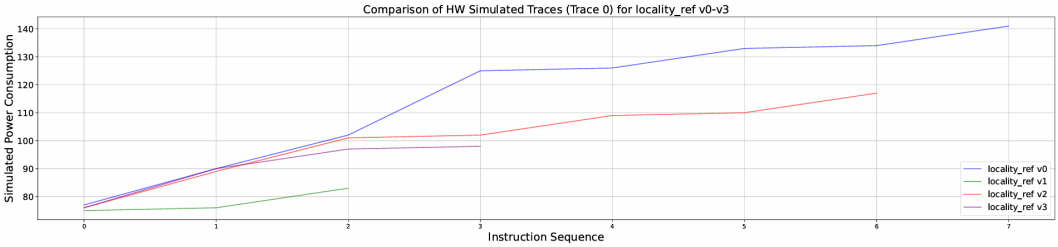

Figure 5.12: Simulated HW Traces in a Single Plot for Locality Refresh

consumption from instruction 0 to 7. Structured mv and xor operations result in registers loading values with increasing Hamming Weights.

Locality Ref v1 is very flat initially; slight rise toward the end. Operations directly modify registers (e.g., xor a0, a0, a2), so fewer transitions occur across distinct registers. This suppresses some temporal leakage but may expose more spatial leakage, especially on registers a0 and a1 reused with sensitive data.

Locality Ref v2 is similar to v0. Despite register renaming, the value transitions (especially via xor) still carry secret-dependent characteristics.

Locality Ref v3 has a gradual power rise over 3 instructions, smaller magnitude compared to others. Irregular update pattern spreads transitions unevenly. This disrupts alignment and magnitude of transitions, making it harder to isolate sensitive changes. This has the lowest apparent leakage under HW model among all versions.

5.3 Correlation-based Analysis

Correlation analysis is a widely used statistical technique for detecting potential side-channel leakages in cryptographic implementations. The core objective is to assess whether there exists a statistically significant relationship between the device's power consumption (or simulated power traces) and certain intermediate values derived from the internal computations of the cryptographic algorithm, such as register contents or masked shares. We've applied this analysis for detecting leakage in our gadgets.

ISW AND Gadget

Figure 5.13 and 5.14 show Hamming Distance (HD) correlation analysis.

For ISW AND v0, correlation values remain within ±0.2, with low amplitude fluctuations. In v1, fluctuations are more frequent and noisy, with some sharp spikes (e.g., near sample 10 and 20). In v2, stronger correlation spikes (up to approximately 0.25) indicate a clear relation between HD and the computed value. This version is leaking significantly.

Figure 5.15 and 5.16 show Hamming Weight (HW) correlation analysis.

In ISW AND v0 or baseline version, correlation values again hover within ±0.1, with a clean and smooth profile. v1 is similar to HD, this variant shows more erratic correlation patterns. The memory activity is likely producing secondary leakage effects, making this version more vulnerable under HW assumptions. v2 has high correlations (approximately 0.2 and above) observed in several time samples. These are clear indicators of leakage, due to direct computation on unmasked sensitive values.

Toffoli Gate Gadget

Figure 5.17 and 5.18 show Hamming Distance (HD) correlation analysis.

Toffoli v0 or baseline shows moderate leakage. There are some visible correlation

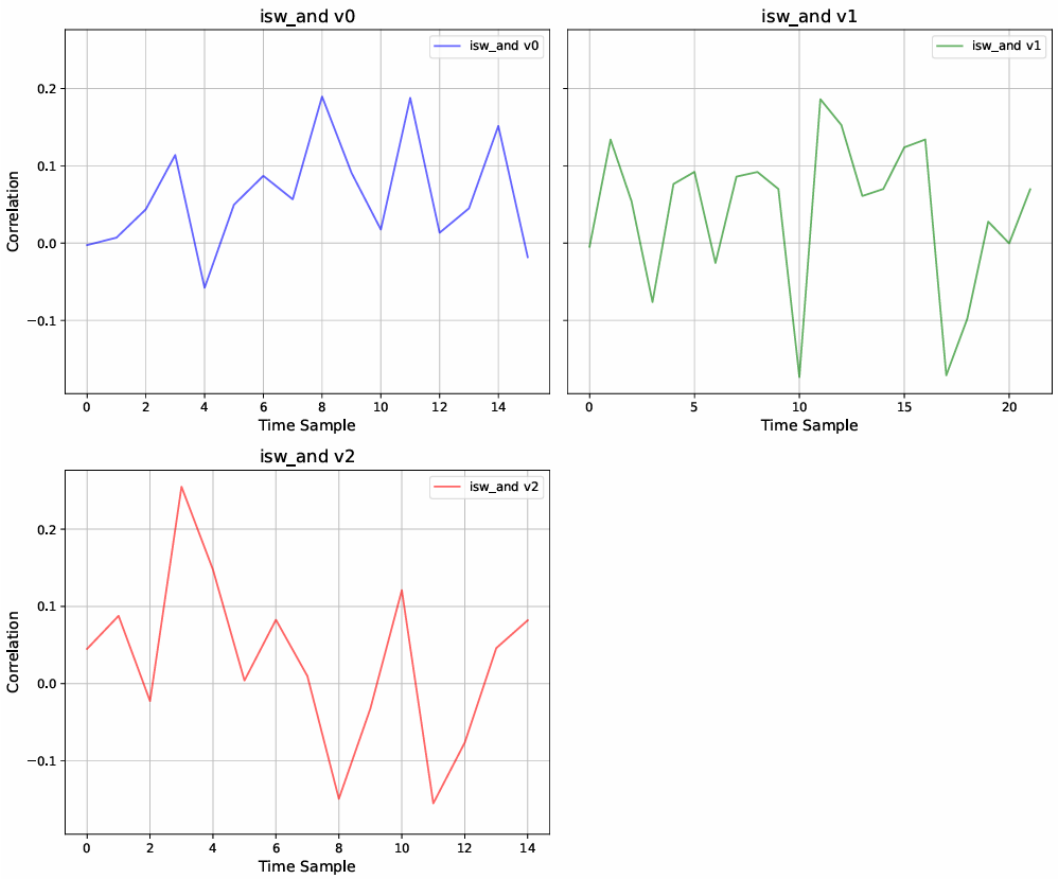

Figure 5.13: Correlation of HD Traces for ISW AND

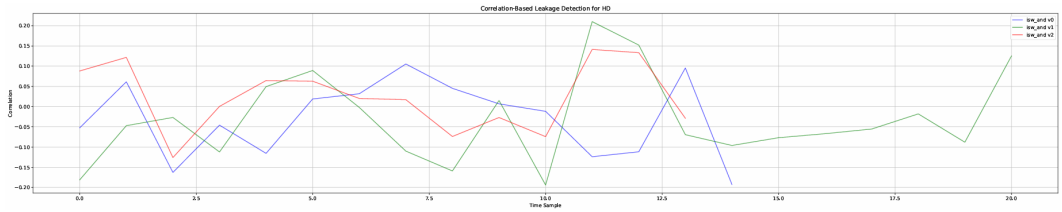

Figure 5.14: Correlation of HD Traces in a Single Plot for ISW AND

peaks around data transitions. v1 has similar leakage as baseline, but peaks occur at shifted time samples due to instruction movement. v2 has high leakage. It changes temporary registers, which causes switching activity, as a result, increasing correlation. v3 also has high leakage. Here shifting order affects when transitions occur, causing distinct spikes. v4 has comparatively low leakage. here added operations introduce noise and disrupt clean switching patterns. v5 has moderate leakage. v6 has low to medium leakage. v7 has the lowest leakage. Here, clearing registers prevents reuse, breaking transition correlations.

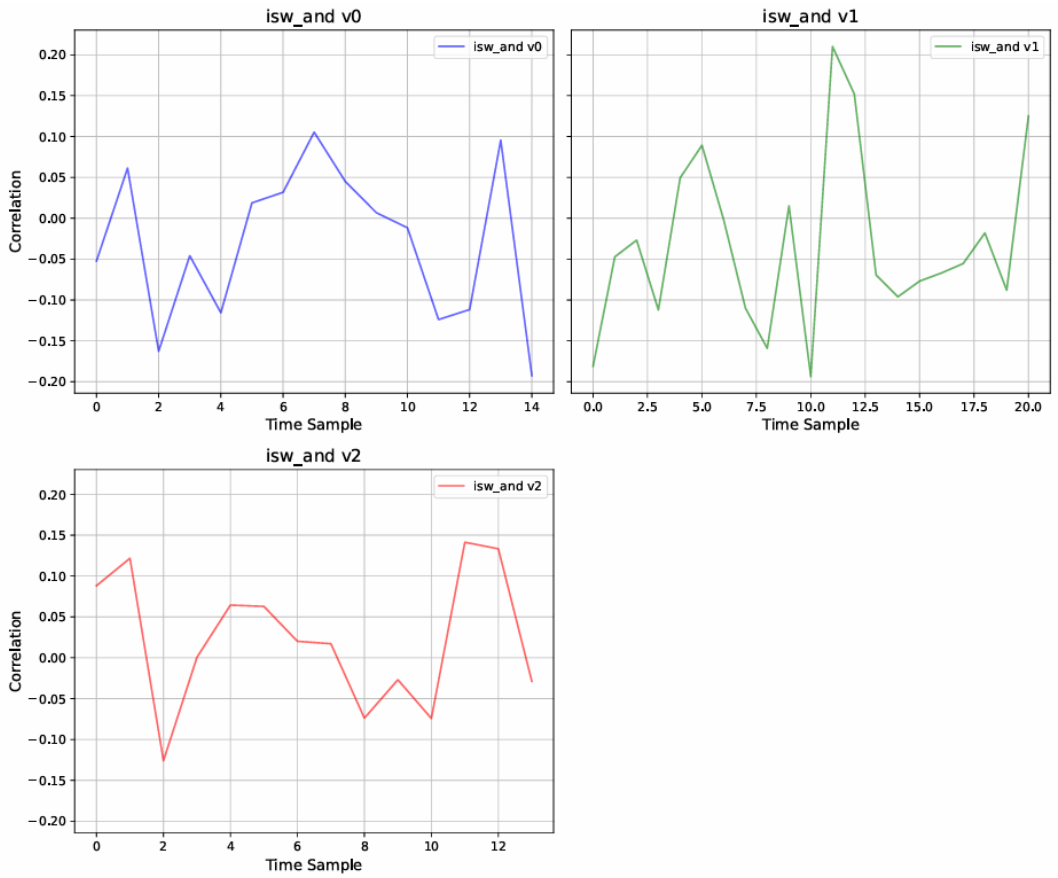

Figure 5.15: Correlation of HW Traces for ISW AND

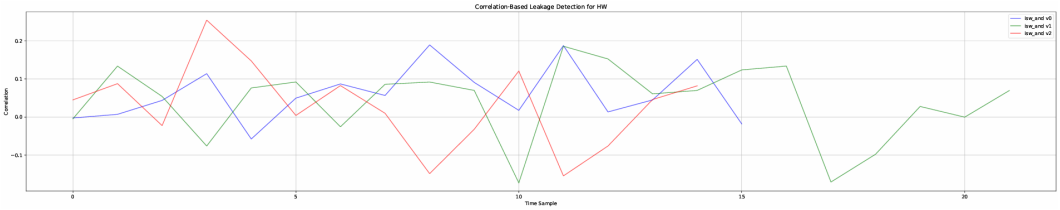

Figure 5.16: Correlation of HW Traces in a Single Plot for ISW AND

Figure 5.19 and 5.20 show Hamming Weight (HW) correlation analysis.

Toffoli gate v0 or baseline shows moderate leakage. HW of register values reveals sensitive information. In v1, the leakage pattern remains, but timing is affected by reordering. There is high leakage for v2. New registers have different HW, exposing sensitive data. v3 also has high leakage. v4 has low leakage. Here masking effect from redundant operations reduces HW correlation. v5 has medium-high leakage. v6 has medium leakage. Here HW correlates with intermediate values. Finally, v7 has the lowest leakage because wiping temporary values removes HW-based leakage

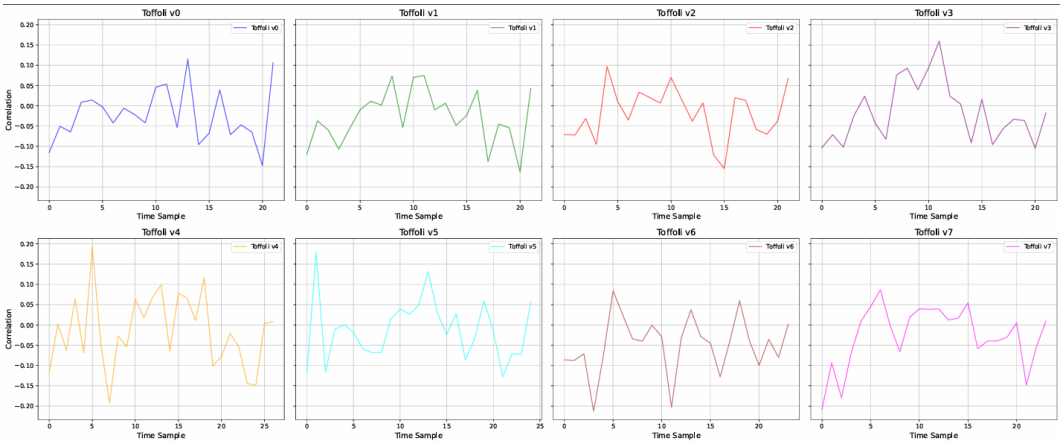

Figure 5.17: Correlation of HD Traces for Toffoli Gate

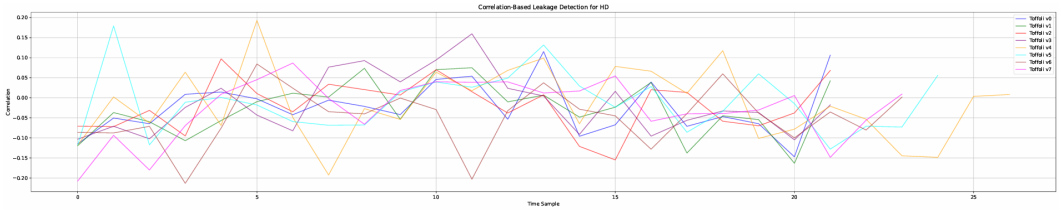

Figure 5.18: Correlation of HD Traces in a Single Plot for Toffoli Gate

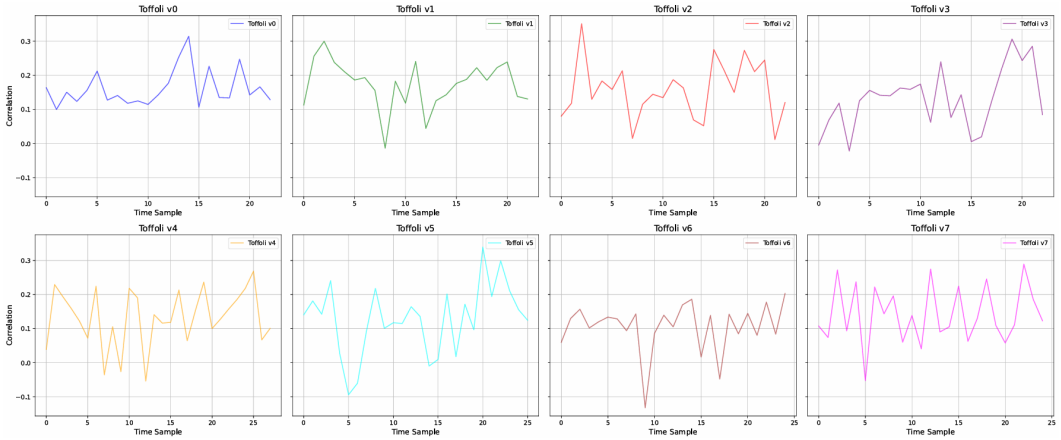

Figure 5.19: Correlation of HW Traces for Toffoli Gate

efficiently.

Locality Refresh Gadget

Figure 5.21 and 5.22 show Hamming Distance (HD) correlation analysis.

Locality Ref v0 or baseline shows noticeable correlation with peaks up to ±0.2. This indicates HD-related leakage. v1 has flat, low-level correlation. v2 has the highest

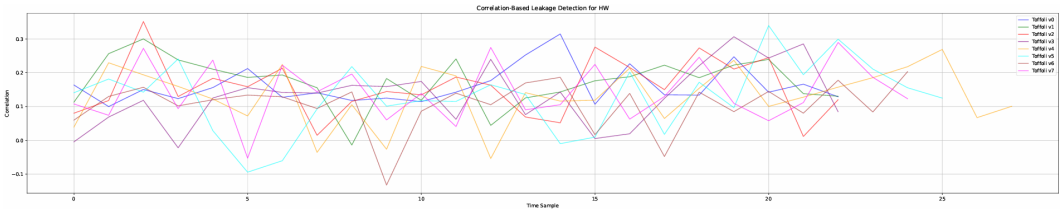

Figure 5.20: Correlation of HW Traces in a Single Plot for Toffoli Gate

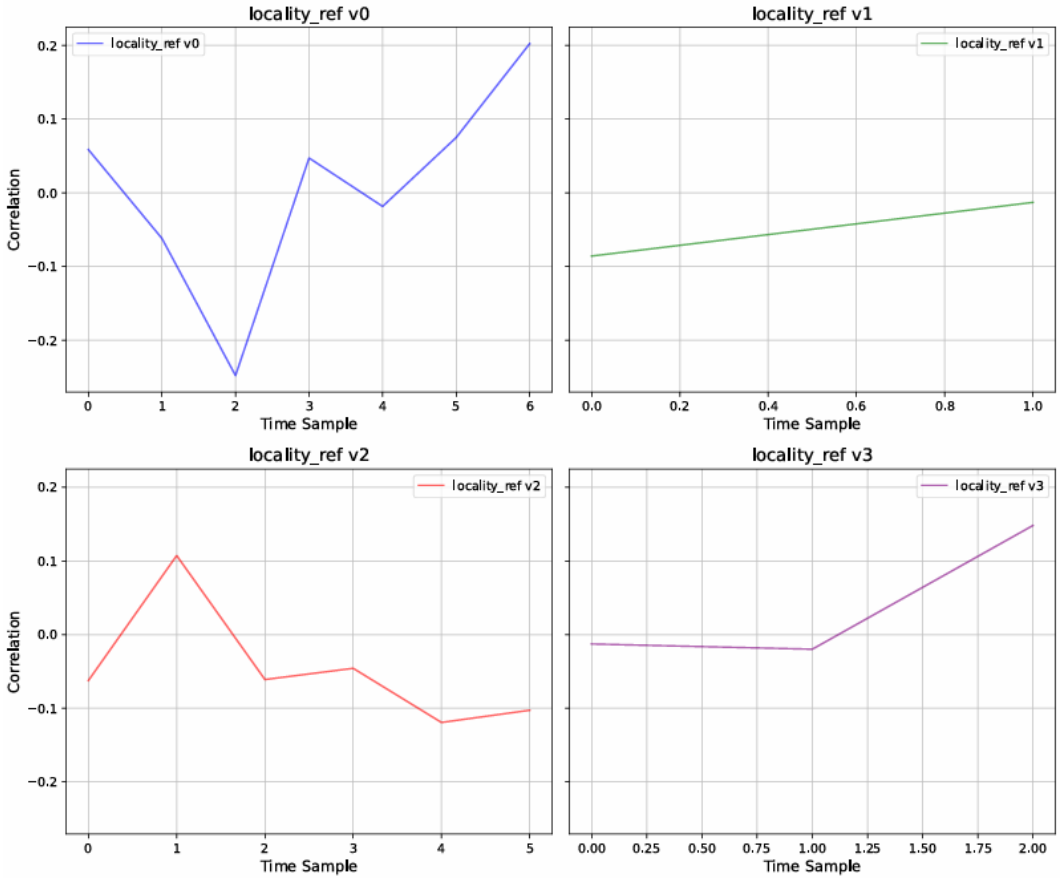

Figure 5.21: Correlation of HD Traces for Locality Refresh

correlation fluctuation among all (up to ±0.2). Major leakage is detected here. v3 has a slight increasing trend, but low values. It has mild leakage.

Figure 5.23 and 5.24 show Hamming Weight (HW) correlation analysis.

Locality Ref v0 or baseline version has fluctuations in correlation with peaks around ±0.1. v1 is almost flat, near zero. It has very low or negligible leakage. v2 has higher variability, ranging between -0.15 to +0.15. Here, leakage is more pronounced. Finally, v3 has weak correlation and low peaks. This is similar to v1.

Figure 5.22: Correlation of HD Traces in a Single Plot for Locality Refresh

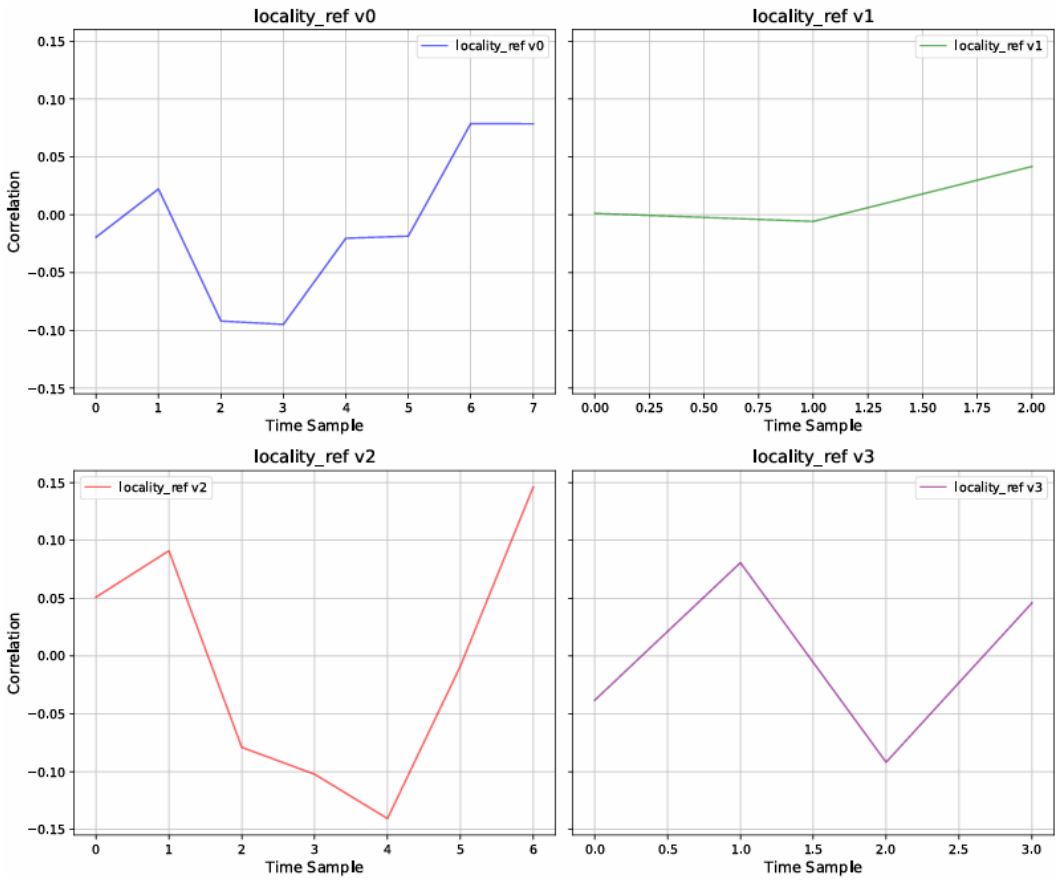

Figure 5.23: Correlation of HW Traces for Locality Refresh

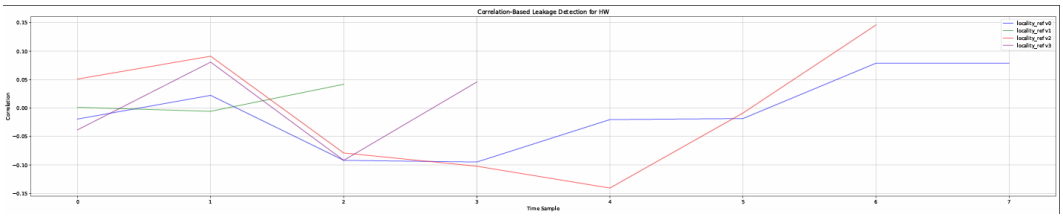

Figure 5.24: Correlation of HW Traces in a Single Plot for Locality Refresh

Chapter 6

DISCUSSIONS

This study systematically evaluated side-channel leakage in masked cryptographic primitives across multiple variants of ISW masked AND, Toffoli-style gadgets, and locality refresh operations, implemented on a lightweight RISC-V core. Leakage was analyzed under both the Hamming Weight (HW) and Hamming Distance (HD) models through correlation-based techniques using real trace measurements.

Across all gadget types, the HD model consistently revealed stronger and more persistent leakage compared to the HW model. This highlights the dominance of bit transitions (i.e., how data changes over time) as the primary source of side-channel leakage on hardware platforms like PicoRV32. Variants that performed well in HW analysis still showed vulnerabilities in HD, demonstrating that a static value-based view of leakage is often insufficient.

The v1 variant was the most secure across all three gadget types, consistently producing low correlation regardless of the leakage model. v2, by contrast, emerged as the leakiest, particularly in the HD domain, suggesting structural weaknesses in the instruction ordering or timing of transitions.

These findings underscore that functionally equivalent implementations can exhibit significantly different leakage behavior, even on simple cores. The choice of register allocation, instruction interleaving, and operand scheduling plays a crucial role in shaping the microarchitectural activity observable through EM traces.

While this phase of the project focused on classical SCA techniques for labeling and understanding leakage profiles, the ultimate goal is to build a high-quality labeled dataset for training and evaluating deep learning-based leakage models. These upcoming efforts will leverage the insights gained from this analysis—especially regarding variant sensitivity and model-specific leakage visibility—to inform dataset structure, input formatting, and model architecture. The findings here form the empirical foundation upon which more advanced, data-driven side-channel evaluations will be constructed in future work.

BIBLIOGRAPHY

[1] John Gaspoz and Siemen Dhooghe. “Threshold implementations in software: Micro-architectural leakages in algorithms”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023), pp. 155–179.

[2] Pantea Kiaei et al. “Saidoyoki: Evaluating side-channel leakage in pre-and post-silicon setting”. In: *Cryptology ePrint Archive* (2021).

[3] Miao He et al. “RTL-PSC: Automated power side-channel leakage assessment at register-transfer level”. In: *2019 IEEE 37th VLSI Test Symposium (VTS)*. IEEE. 2019, pp. 1–6.

[4] Amisha Srivastava et al. “SCAR: Power Side-Channel Analysis at RTL Level”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2024).