Root-cause Analysis of the Side Channel Leakage from ASCON Implementations

Zhenyuan Liu and Patrick Schaumont$^{[0000-0002-4586-5476]}$

Worcester Polytechnic Institute, Worcester MA 01609, USA

Abstract. Root-Cause Analysis of Side Channel Leakage (SCL) aims to identify the components of observed SCL. We analyze ASCON's SCL for two variants implemented on a RISC-V SoC: a hardware coprocessor with an iterated implementation of ASCON-128, and a software implementation of ASCON-128 on RISC-V (RV32IMC). We use gate-level power simulation to find the power traces, and to identify which portions of the hardware/software implementation show most leakage of the secret key. Our analysis breaks out the leakage according to the major processing phases of ASCON and according to their implementation. We also compare the simulated traces with measurements collected from a 180nm ASIC implementation of the same design[1].

Keywords: Power-based Side-channel Leakage · Gate-level Simulation · Pre-silicon Side-channel Leakage Analysis

1 Introduction

The implementation of a new cryptographic design mandates side-channel leakage assessment [PGA+22] to evaluate the extent of side-channel vulnerability, typically by means of a standard testing mechanism [ISO16]. Side-channel leakage assessment also helps to identify and develop leakage models for side-channel analysis, and to test the quality of countermeasures used in the implementation. In this paper, we specifically discuss power-based side-channel leakage. We use the abbreviation SCL to mean power-based side-channel leakage.

In contrast to side-channel leakage assessment, *root-cause analysis* of SCL aims to explain how and where the leakage occurs in a design. Root-case analysis demonstrates a property of the implementation, rather than a property of the SCL. This type of analysis helps the designer to understand what parts of the implementation cause side-channel leakage, which is a starting point for the development of countermeasures, or for debugging of side-channel leakage related defects.

Root-cause analysis uses simulated or measured SCL, as long as one can establish a connection between side-channel characteristics and the components

[1] The research for this paper was supported in part by the National Science Foundation Award 1931639

of the implementation [BBYS21]. The terms *pre-silicon* and *post-silicon* evaluation have been used to distinguish simulation-based methods from measurement-based methods. For example, ABBY builds a micro-architectural leakage model from measurements taken from an ARM M0 core [BIBB21], while MAPS builds such micro-architectural model from simulated register values for an ARM M3 [LCGD18]. A comprehensive list of recently proposed pre- and post-silicon tools for side-channel leakage assessment as well as for root-cause analysis may be found on `https://ileanabuhan.github.io/Tools/`. While side-channel leakage assessment only aims to show a dependency between secret data and observed SCL, root-cause analysis must also attribute the observed SCL to a specific component. The granularity of these components is determined by the power modeling abstraction level, and varies from a software instruction down to a logic gate.

In traditional CMOS technology, power is directly related to the logic transitions in the design, and side-channel leakage is the result of the dependency between a secret value and these transitions. The root-cause analysis then becomes a matter of tracking the component that has directly caused a given data transition, for example tracking logic gate that flips a given wire in a netlist. However, not every data transition may be available in the power model at the selected abstraction level. At register-transfer level or at instruction-set level, for example, only transitions in select registers are visible, while those in the datapath remain invisible [LCGD18,HPN+19]. The SCL of such abstracted models will not cover every possible source, and this will also limit the accuracy of the root-cause analysis. Research has also pointed out that data transitions are not the only source of SCL. Other circuit-level effects such as glitches, static power dissipation [MMR20], coupling [PRP+19], and signal integrity on the power delivery [GS20] all have been identified as contributing to power-based side-channel leakage. Furthermore, even a highly accurate power model of on-chip power dissipation may still be insufficient to fully explain the root cause, as it has been shown that the off-chip components such as decoupling capacitors can affect the SCL [GGB+23]. Hence, the present state of knowledge in SCL simulation does not provide sufficiently accurate power models that guarantee the absence of false negatives: the absence of side-channel leakage in simulation does not imply the absence of side-channel leakage in the implementation. However, we do have power models that ensure, with high confidence, the absence of false positives: the presence of side-channel leakage in the simulation will also lead to side-channel leakage in the implementation. In this paper, we use gate-level power estimation because it provides a decent trade-off between power model detail and simulation performance [YSW+20].

Architecture Correlation Analysis Our root-cause analysis method is called *Architecture Correlation Analysis* (ACA) [KYL+22]. The basic flow of ACA is shown in Figure 1. For a given set of test vectors, the ACA flow will perform logic simulation followed by high-resolution power simulation. Each test vector yields a trace reflecting the power consumption of the design while processing the test vector. ACA will then compute a *leakage time interval* (LTI) over the traces,

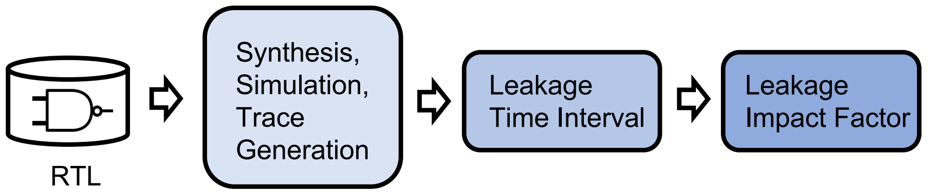

Fig. 1. ACA Flow

which corresponds to the time windows within the ensemble of side-channel leakage traces that are considered *leaky*, i.e. at what times the ensemble of traces would be identified as SCL by a side-channel leakage assessment method. The LTI is computed through a global (overall) side-channel leakage assessment. Because traces are in sampled form, the time windows of the LTI are captured as a collection of *leaky points*. For each leaky point, ACA will then rank the logic gates of the design according to the amount of contributed leakage per gate by computing a *leakage impact factor* (LIF) for each gate. The LIF represents the similarity of an individual gate's activity to the overall SCL metric. A higher LIF indicates a gate that has a stronger similarity and thus is more likely to contribute to the overall side-channel leakage of the design. The collection of gates with a LIF higher than a predefined threshold is the *root-cause set* S of SCL. The S set depends on the specific activity of a design, and thus can change per leaky point. Hence, a complete root-cause description of SCL over N leaky points is list of sets $\{S_0, S_1, ..., S_{N-1}\}$. For a detailed discussion on ACA we refer to Kiaei *et al.* [KYL+22].

Related work in ASCON SCL assessment In this paper, we perform the SCL root-cause analysis of ASCON implementations. Compared to substitution permutation network (SPN) algorithms such as AES, the side-channel leakage from sponge constructions is less well investigated. A few authors have presented side-channel leakage assessment for hardware implementations of ASCON [SD17] [RAD20][DCBG+17][GWDE17], and for software implementations [AFM18][SS23]. These attacks target S-box output in the ASCON initialization phase and they require several thousand traces (ASCON iterations) for success. In the context of the NIST Light Weight Cryptography competition, GMU has organized a comprehensive benchmarking effort for the side-channel leakage assessment of the finalists including a protected hardware implementation of ASCON [MBA+23]. To our knowledge, no earlier work has presented a root-cause analysis of ASCON, i.e. explains how side-channel leakage is distributed over individual gates (for hardware implementations) or over individual software instructions (for software implementations).

Contributions and Outline The contributions of our work are as follows.

1. We present a RISC-V based system-on-chip (SoC) architecture with a hardware and a software implementation of ASCON128, and use a design flow that enables simulation as well as measurement of side-channel leakage.
2. We use a non-specific root-cause analysis over specific regions of ASCON AEAD processing that directly manipulate the key. We do this for both the hardware implementation as well as for the software implementation. Selected regions of interest include the coprocessor key-loading process, the ASCON first-round initialization phase (for hardware and software), and the ASCON finalization phase (for software).
3. For each region of interest we determine the root-cause set of leaky gates as a function of the processing stage of ASCON (initialization, associated-data processing, plaintext encryption, and finalization). When analyzing ASCON software, we express the leakage pattern as a root-cause set of leaky instructions by back-annotating leaky gates and cycles to software [KS22].

The remainder of this paper is organized as follows. We first describe the main properties of our implementation in Section 2. Next, we perform root-cause analysis on the ASCON hardware coprocessor in Section 3, as well as root-cause analysis on the ASCON RISCV software implementation in Section 4. Finally, in Section 5 we validate our simulation results by comparing a side-channel leakage assessment on the gate-level simulation with a side-channel leakage assessment of the ASIC implementation. We conclude in Section 6.

2 ASCON Coprocessor and ASCON Software

We analyze two different implementations of ASCON, a first one running as a hardware coprocessor and a second one executing as software. Both ASCON implementations are implemented on a small RISC-V SoC in 180nm standard cells (Figure 2, top). The chip executes bare-metal software from on-chip RAM. The complexity of the chip design is captured in Table 1.

– The software implementation of ASCON is based on the reference implementation for ASCON128v12 by the ASCON Designers [asc23], compiled with O2 optimization level. We refer to this target as SWASCON-128.
– The hardware implementation of ASCON is an iterated design based on M. Fivez's open-source implementation [Fiv16]. This kernel is encapsulated in a memory-mapped interface and integrated in a PicoRV32 RISC-V based SoC (Figure 2, bottom). We refer to this target as HWASCON-128. The fundamental characteristic of a memory-mapped interface is that all communication to the coprocessor hardware – data as well as control signals – are expressed as memory writes/reads in a shared address space between the software and the hardware. HWASCON uses 18 shared addresses to communicate the secret key value, the associate data, the public nonce, the plaintext, the ciphertext and the tag. The coprocessor maintains a memory-mapped status register that is used by the software to determine if a given

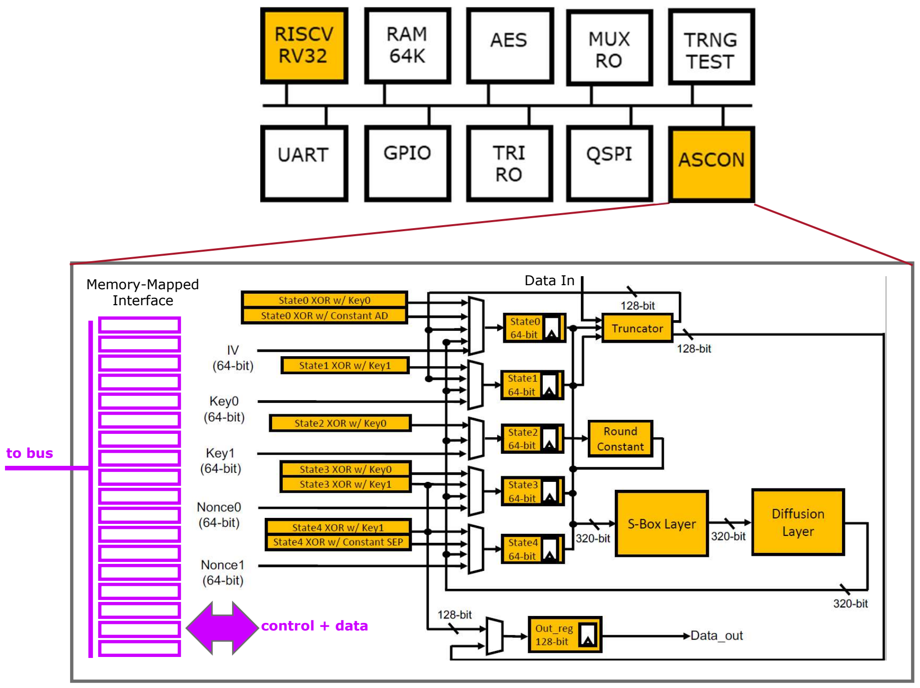

Fig. 2. Block Diagram of PicoSoc with ASCON Coprocessor

coprocessor transfer (such as a block of Associated Data) has been fully processed.

Trace Collection To collect power traces from the simulation of the chip, we use gate-level simulation and Cadence Joules to capture both static and dynamic power consumption. Cadence Joules is limited to traces of 1,000 *frames*, where frame captures the average power over a small time window in the simulation. To limit the overhead of data collection, we use a software trigger that marks the beginning and end of the region of interest for analysis. The software trigger toggles a GPIO pin, which can be detected in the simulation testbench to enable the event collection process. To collect power traces from the chip measurement, we use Chipwhisperer Husky which synchronously clocks the chip while recording the power trace. Chipwhisperer Husky enables traces of up to 131,072 points. The same software trigger is used to start and stop recording of a trace.

Non-specific Leakage Test An important design choice in our root-cause analysis, is the selection of a leakage metric. We are using a *non-specific* leakage test, which measures the statistical difference between two groups of traces. Group 0 are traces generated for a random secret key, while group 1 are traces generated for a constant secret key. The test statistic for leakage is either the Welch-t t-value between group 0 and group 1, or Pearson's correlation with the group

Table 1. Chip Characteristics

Technology	TSMC 180nm (tcb018gbwp7t_270a)
Comb. Cells	43,724 cells
Seq. Cells	13,947 cells
Core Area	10.69 sq mm
Chip Area	13.35 sq mm
SRAM	64 KByte (8 x 8K macro)

identifier (+1 for group 0 and -1 for group 1). The ranking process of ACA will attribute a higher LIF (Leakage Impact Factor) to gates with a higher t-value or with a higher correlation value [KYL$^+$22].

3 Root-cause Analysis of ASCON Hardware SCL

In this experiment, our objective is to present the root-cause analysis of key-dependent SCL in ASCON Coprocessor on RISC-V Soc. We narrow down to two regions that has direct interaction with the 128-bit key as our analysis targets:

1. ASCON Coprocessor Key-Load.
2. ASCON Coprocessor Initialization.

To maximize the proportion of key-dependent SCL in the overall power consumption of ASCON Coprocessor on RISC-V Soc, we apply 2000 test vectors consisting of random vs. fixed key (1000 test vectors each group) with constant nonce, associated data, and plaintext. The clock frequency is 4MHz, and the power simulation is oversampled at 4 samples per clock cycle.

For root-cause analysis, we present ACA LTI to identify the leaky clock cycles over simulation time. To quantify the key-dependent leaky gates over clock cycles, we rank the logic gates of the design per clock cycle according to the amount of contributed leakage per gate by computing LIF. Moreover, we express the leakage pattern as a root-cause set of leaky instructions by back-annotating leaky gates and cycles to software.

ASCON Coprocessor Key-Load The operation of the ASCON Coprocessor involves software as well as hardware. To initialize the coprocessor, the software will load the key, associated data and plaintext from RAM to the hardware coprocessor. While this operation strictly does not involve ASCON computation, it is included in every execution of the ASCON coprocessor; the key can be reused over multiple operations but must be loaded at least once. For this reason, we also investigate the key loading process.

Listing 1 summarizes the software operations involves in a key load. It takes four instructions (lw, lw, addi, sw) to load a 32-bit key[0] from RAM to coprocessor (Instr. 70e, 712, 716, 718). This is repeated four times to complete a 128-bit key load using a 32-bit bus interface. After a 128-bit key is loaded, the software writes a key update control signal to the coprocessor (Instr. 744), and wait for the coprocessor to send back a key ready signal (Instr. 754). Finally, the software sends a key valid signal to complete the sequence (Instr. 75c).

Listing 1. Coprocessor Key-Load Sequence. Marked addresses refer to Fig. 3's X-axis

```
70e  lw   a4,-228(s0) # load key[0] from RAM
712: lw   a5,-20(s0)
716: addi a5,a5,4
718  sw   a4,0(a5)    # store key[0] to coproc
71a  lw   a4,-224(s0) # load key[1] from RAM
71e: lw   a5,-20(s0)
722: addi a5,a5,8
724  sw   a4,0(a5)    # store key[1] to coproc
726  lw   a4,-220(s0) # load key[2] from RAM
72a: lw   a5,-20(s0)
72e: addi a5,a5,12
730  sw   a4,0(a5)    # store key[2] to coproc
732  lw   a4,-216(s0) # load key[3] from RAM
736: lw   a5,-20(s0)
73a: addi a5,a5,16
73c  sw   a4,0(a5)    # store key[3] to coproc
73e: lw   a5,-20(s0)
742: li   a4,4
744: sw   a4,0(a5)    # coproc control: key update
746: nop
748: lw   a5,-20(s0)
74c: addi a5,a5,68
750: lw   a5,0(a5)
752: andi a5,a5,1
754: beqz a5,748      # wait for key update ready
756: lw   a5,-20(s0)
75a: li   a4,8
75c  sw   a4,0(a5)    # coproc control: key valid
```

The entire ASCON Coprocessor Key-Load process requires 214 clock cycles to complete. We identify the LTI with a $\rho_{threshold}$ of 0.2, which flags 198 clock cycles out of these 214 clock cycles as containing potentially leaky samples. Figure 3 shows the root-cause analysis results from ACA. The top figure shows the correlation of the trace partitions with the group identifier. The bottom figure shows the number of leaky gates over time, indicating each gate's source code (RTL module) in color (red indicates a leaky gate from the PicoRV, blue indicates a leaky gate from the ASCON coprocessor, and black indicates the rest of the modules from Figure 2 (top)). We observe a clear leaky-gate transiting pattern over time for each load key: flagging 750 leaky gates from the PicoRV at the beginning, and 100 leaky gates from the coprocessor at the end. This pattern validates the correctness of our software Key-Load from the RTL-level as well as shows how detailed that ACA can perform in terms of the root-cause analysis on SCL.

We also observe that a high correlation value (Figure 3 (top)) does not imply a high number of leaky gates (Figure 3 (bottom)). This is caused by the noiseless simulation: even when there are almost no activities in the design (such as in the waiting loop at Instr. 754), a high correlation can still exist. For this reason, it is important to simultaneously consider Figure 3 top and bottom. A high correlation value will only have impact on the overall side-channel leakage when

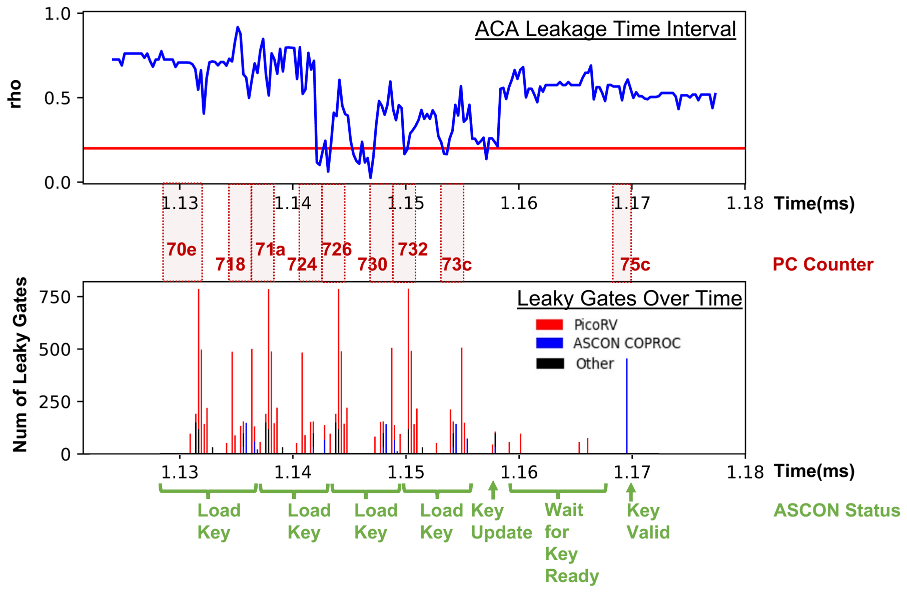

Fig. 3. ACA LTI Results (top) and Num of Leaky Gates over Simulation Time (bottom) on HWASCON-128 Co-Processor Key-Load Phase with PC counter values in Red and ASCON Program Status in Green

a high number of leaky gates are involved. Furthermore, we note that 750 gates are still a tiny fraction of the overall 57,671 cells in the chip.

By annotating the time instant of the peaks in the leaky gates to the program counter of the PC, we can back-annotate the leakage to activities in the software. This way, we can find 9 instructions (highlighted in red in Listing 1 and Figure 3's PC Counter) that are responsible for a significant amount of leaky gates (key-dependent SCL). These instructions copy the key value from RAM to a coprocessor register. There is another peak in Figure 3, which maps to instruction 75c in Listing 1. Even though this instruction is a control instruction for the coprocessor, its effect is to move the key value from the register in the memory-mapped interface to a register in the ASCON hardware core. This creates additional side-channel leakage, and the source of the side-channel leakage comes from the ASCON coprocessor.

ASCON Coprocessor Initialization Moving on to the root-cause analysis inside of the ASCON coprocessor, this selected region covers the entire ASCON Coprocessor Initialization phase and software control writes. Listing 2 summaries its software instructions. 12-round of coprocessor initialization happens between sends nonce valid (Inst. 7a6) and loads AD[0] from RAM (Inst. 7b0). The entire ASCON Coprocessor INIT simulation takes 198 clock cycles to complete. We identify the LTI with a $\rho_{threshold}$ of 0.2, which flags 185 clock cycles out of the 198 clock cycles as containing potentially leaky samples. Figure 4 shows the root-

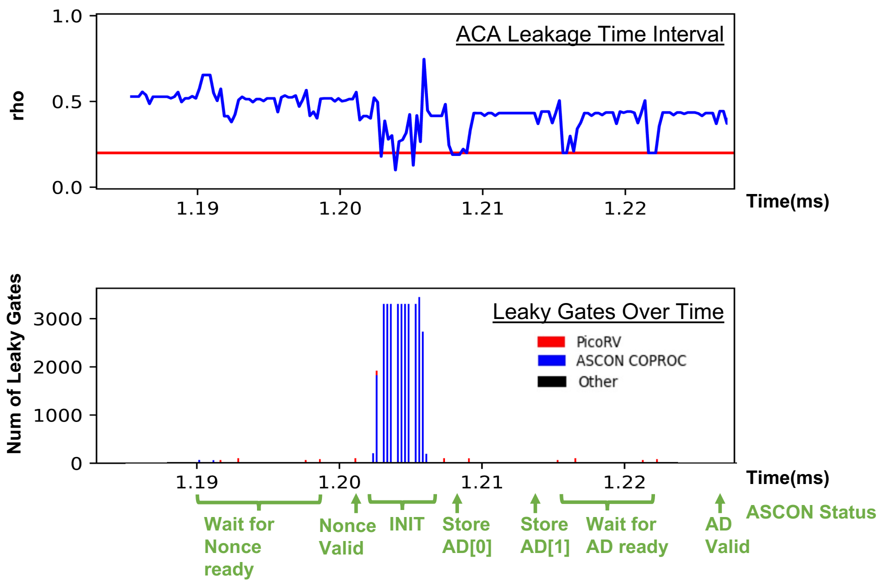

Fig. 4. ACA LTI Results (top) and Num of Leaky Gates over Simulation Time (bottom) on HWASCON-128 Co-Processor Initialization Phase with PC counter values in Red and ASCON Program Status in Green

cause analysis results from ACA. We observe that there is no significant amount of key-dependent SCL showing in software instructions, nor from the PicoRV module. This is expected because during software load/store nonce and associate data, there is no sensitive data (key) being processed, and the computations are happening inside of the ASCON coprocessor. For this reason, we see significant amount of key-dependent leaky gates during the coprocessor initialization stage from Figure 4 (bottom). There are 12 clock cycles being flagged over 2000 leaky gates per clock cycle. These 12 clock cycles consist of 12-round of permutation, and the key is directly feed into the initialization, therefore, shows significant amount of key-dependent SCL. Furthermore, we note that 2000 gates are still a tiny fraction of the overall 57,671 cells in the chip.

Listing 2. Coprocessor INIT sequence. Refer to Fig. 4

```
792: lw   a5,-20(s0)  # load ascon co-processor base address
796: addi a5,a5,68
79a: lw   a5,0(a5)
79c: andi a5,a5,2
79e: beqz a5,792      # loop wait for coproc nonce ready
7a0: lw   a5,-20(s0)
7a4: li   a4,16
7a6: sw   a4,0(a5)    # send nonce valid
7a8: lw   a5,-20(s0)
7ac: addi a5,a5,36
7b0: lw   a4,-136(s0) # load AD[0] from RAM
7b4: sw   a4,0(a5)    # store AD[0] to coproc
```

```
7b6: lw   a5,-20(s0)
7ba: addi a5,a5,40
7be: lw   a4,-132(s0) # load AD[1] from RAM
7c2: sw   a4,0(a5)    # store AD[1] to coproc
7c4: nop
7c6: lw   a5,-20(s0)
7ca: addi a5,a5,68
7ce: lw   a5,0(a5)
7d0: andi a5,a5,4
7d2: beqz a5,7c6      # loop wait for coproc AD ready
7d4: lw   a5,-20(s0)
7d8: lui  a4,0x1
7da: addi a4,a4,32
7de: sw   a4,0(a5)    # coproc control: AD valid | AD last block
```

4 Root-cause Analysis of ASCON Software SCL

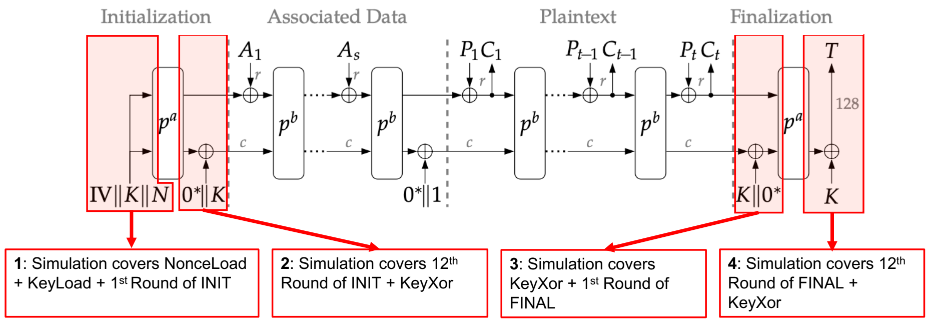
Initialization
Associated Data
Plaintext
Finalization
A_1
A_s
$P_1 C_1$
$P_{t-1} C_{t-1}$
$P_t C_t$
T
p^a
p^b
r
c
128
$IV\|K\|N$
$0^*\|K$
$0^*\|1$
$K\|0^*$
K
1: Simulation covers NonceLoad + KeyLoad + 1st Round of INIT
2: Simulation covers 12th Round of INIT + KeyXor
3: Simulation covers KeyXor + 1st Round of FINAL
4: Simulation covers 12th Round of FINAL + KeyXor

Fig. 5. SWASCON-128 Simulation Experiment Rationale

In this experiment, our objective is to present the root-cause analysis of key-dependent SCL in SWASCON SCL on RISC-V (RC32IMC). Figure 5 shows SWASCON encryption details from Initialization, Associated Data, Plaintext, to Finalization. To select our analysis regions, we narrow down to four regions that has direct interaction with the 128-bit key in SWASCON as our targets:

1. SWASCON Nonce-Load, Key-Load, and 1st Round of Initialization.
2. SWASCON 12th Round of Initialization and Key-XOR.
3. SWASCON Key-XOR and 1st Round of Finalization.
4. SWASCON 12th Round of Finalization and Key-XOR.

To maximize the proportion of key-dependent SCL in the overall power consumption of SWASCON on RISC-V (RC32IMC), we apply 1000 test vectors consisting of random vs. fixed key (500 test vectors each group) with random nonce, constant associated data, and constant plaintext. The clock frequency is 4MHz, and the power simulation is sampled at 1 sample per clock cycle.

For root-cause analysis, we use the same method as in section 3. We present ACA LTI to identify the leaky clock cycles over simulation time. To quantify the key-dependent leaky gates over clock cycles, we rank the logic gates of the design per clock cycle according to the amount of contributed leakage per gate by computing LIF. Moreover, we express the leakage pattern as a root-cause set of leaky instructions by back-annotating leaky gates and cycles to software.

SWASCON Nonce-Load, Key-Load, and 1st Round of Initialization Comparing to ASCON Coprocessor, all the SWASCON computations happen in side of the PicoRV. From section 3, we observe that the coprocessor takes one clock cycle on one round of permutation, however, this is not the case in SWASCON. In this selected region, SWASCON takes around 900 clock cycles to complete one round of permutation. Together with Nonce-Load and Key-Load, the entire simulation takes 957 clock cycles to finish. The corresponding software instructions are showing in Listing 3, including load/store nonce and key, and part of the 1st round of permutation (Round Constant, KECCAK SBOX and Linear Diffusion Layer).

Listing 3. Software ASCON Nonce-Load, Key-Load, and 1st Round of INIT. Refer to Fig. 6's X-axis

```
2a68: addi a0,sp,88
2a6a: sw s7,112(sp)    # store nonce[0]
2a6c: sw s3,116(sp)    # store nonce[1]
2a6e: sw s11,124(sp)   # store nonce[2]
2a70: sw a1,120(sp)    # store nonce[3]
2a72  sw s4,96(sp)     # store key[0]
2a74  sw s6,100(sp)    # store key[1]
2a76  sw s10,104(sp)   # store key[2]
2a78  sw s0,108(sp)    # store key[3]
2a7a: jal ra,fe4 <P12>

00000fe4 <P12>:
...
fee   lw s1,8(a0)      # load key[0]
ff0   lw ra,12(a0)     # load key[1]
...
1004  lw a5,16(a0)     # load key[2]
1006  lw t4,20(a0)     # load key[3]
...
1026  xori s4,a5,240   # round constant: state_reg[2] XOR with 0xf0
...
1036  not a7,t4        # not key[3]
103a  xori a5,a5,-241  # XORI key[2]
...
1246: jal ra,19c
```

We identify the LTI with a $\rho_{threshold}$ of 0.2, which flags 239 clock cycles out of the 957 clock cycles as containing potentially leaky samples. Figure 6 shows the root-cause analysis results from ACA. By annotating the time instant of the peaks in the leaky gates to the program counter of the PC, we find 11 instructions (highlighted in red in Listing 3 and Figure 6's PC Counter) that are responsible for a significant amount of leaky gates (key-dependent SCL). These instructions show direct interaction with the sensitive data (key), including four

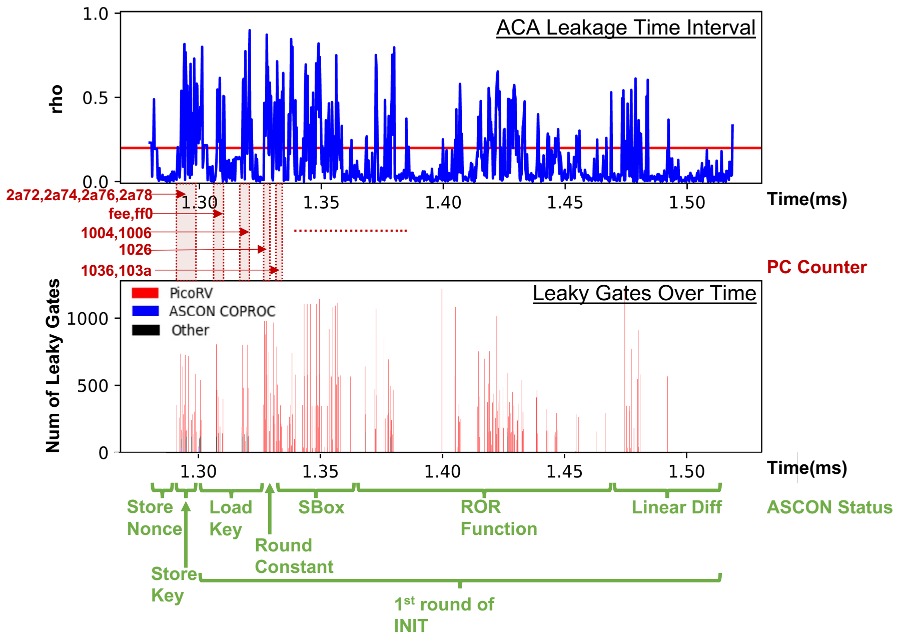

Fig. 6. ACA LTI Results (top) and Num of Leaky Gates over Simulation Time (bottom) on SWASCON-128 Nonce-Load, Key-Load, and 1st Round of Initialization Phase with PC counter values in Red and ASCON Program Status in Green

load/store key instructions, Round Constant (Instr. 1026), and more for the rest of the permutation.

We observe that the key is being used multiple times in the software instructions causing large amount of leaky gates being flagged in Figure 6, and majority of these leaky gates come from the PicoRV module, but still a tiny fraction of the overall 57,671 cells in the chip. Moreover, we notice that there is a differences between SWASCON and ASCON coprocessor during the initialization phase. In ASCON coprocessor, each round of permutation finishes within one clock cycle, therefore, we see 12 clock cycles with significant amount of leaky gates being flagged in Figure 4 (bottom). On the other hand, SWASCON takes around 900 clock cycles to complete one round of permutation, so when there are no key-dependent instructions executing, the number of leaky gates being flagged in that clock cycle is close to zero. We also observe that half passed the Linear Diffusion Layer from the 1st round of permutation, the number of leaky gates (key-dependent SCL) significantly reduces in SWASCON, which is not very obvious from the coprocessor experiment results (Figure 4 (bottom)). This shows that in SWASCON, the sensitive data (key) is gradually being consumed over clock cycles, flagging less key-dependent leaky gates towards the end of the 1st round of permutation.

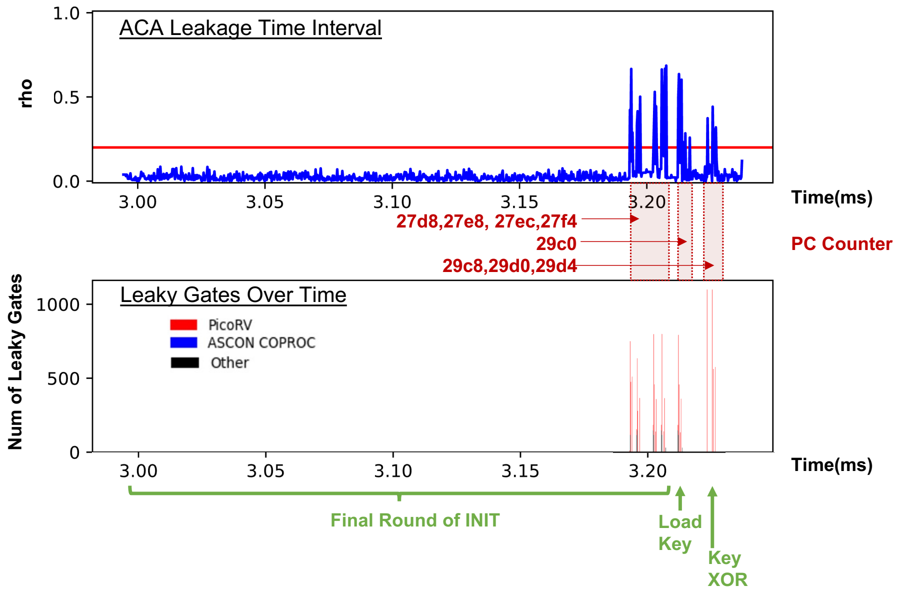

Fig. 7. ACA LTI Results (top) and Num of Leaky Gates over Simulation Time (bottom) on SWASCON-128 12th Round of Initialization and Key-XOR Phase with PC counter values in Red and ASCON Program Status in Green

SWASCON 12th Round of INIT and Key-XOR Moving on to the root-cause analysis at the end of the SWASCON initialization phase. During the 12th round of initialization and Key-XOR, the entire simulation takes 973 clock cycles to complete. We identify the LTI with a $\rho_{threshold}$ of 0.2, which only flags 30 clock cycles out of the 973 clock cycles as containing potentially leaky samples. Figure 7 shows the root-cause analysis results from ACA. We find 8 instructions (highlighted in red in Listing 4 and Figure 7's PC Counter) that are responsible for a significant amount of leaky gates (key-dependent SCL), but still a tiny fraction of the overall 57,671 cells in the chip. We only see a few clock cycles marked as leaky at the end of permutation and this is expected. Because at the end of the 12th round permutation, it loads the final state registers values (Instr. 27d8, 27e8, 27ec, 27f4) and a 128-bit key (Instr. 29c0), and perform the Key-XOR operation (Instr. 29c8, 29d0, 29d4). Only one out of four key load instruction, and three out of four xor instructions are flagged with significant amount of leaky gates. This shows us that the post-permutation Key-XOR during the initialization phase contains less key-dependent SCL even though the key is directly being used in the computation.

Listing 4. Software ASCON 12th Round of INIT and Key-XOR. Refer to Fig. 7's X-axis

```
00000fe4 <P12>:
...
```

```
27d8 lw   s0,88(sp)   # load state_reg
...
27e8 lw   s4,72(sp)   # load state_reg
...
27ec lw   s6,64(sp)   # load state_reg
...
27f4 lw   s10,48(sp)  # load state_reg

27fa: ret

29bc: jal  ra,fe4 <P12>
29c0 lw   t0,124(sp)  # load key
29c2 lw   s3,112(sp)   # load key
29c4 lw   s7,116(sp)   # load key
29c6 lw   s11,120(sp)  # load key
29c8 xor  s8,t0,s0    # xor state_reg and key
29cc xor  s3,s3,s4     # xor state_reg and key
29d0 xor  s7,s7,s6    # xor state_reg and key
29d4 xor  s11,s11,s10 # xor state_reg and key
```

SWASCON Key-XOR and 1st round of FINAL Moving on to the root-cause analysis in the finalization phase. During the Key-XOR and 1st round of finalization, the entire simulation takes 941 clock cycles to finish. We identify the LTI with a $\rho_{threshold}$ of 0.2, which only flags 48 clock cycles out of the 941 clock cycles as containing potentially leaky samples. Figure 8 shows the root-cause analysis results from ACA. We find 14 instructions (highlighted in red in Listing 5 and Figure 8's PC Counter) that are responsible for a significant amount of leaky gates (key-dependent SCL), but still a tiny fraction of the overall 57,671 cells in the chip. We observe that at the beginning of this region, there are significant amount of leaky gates flagged. This is where the Key-XOR happens. Leaky instructions are Key-XOR (Instr. 452e, 4532, 4538, 453c) and store the XORed values (Instr. 4540, 4542, 4544, 4546). We only see few clock cycles marked as leaky during the 1st round of finalization, these are store instructions which store the key-dependent intermediate values when executing KECCAK SBOX and Linear Diffusion layer. Comparing to the 1st round of initialization, the 1st round of finalization flagged way less leaky cycles and gates.

Listing 5. Software ASCON Key-XOR and 1st Round of FINAL. Refer to Fig. 8's X-axis

```
452e xor a5,s5,s0 # key xor
4532 xor t5,s10,s2 # key xor
...
4538 xor t2,s6,s1 #key xor
453c xor ra,s7,s8 #key xor
4540 sw a5,96(sp) # store xored value
4542 sw t5,104(sp) # store xored value
4544 sw t2,100(sp) # store xored value
4546 sw ra,108(sp) # store xored value
4548 jal ra,fe4 <P12>

00000fe4 <P12>:
```

```
...
10bc  sw s5,68(sp)
...
10d0  sw s9,52(sp)
...
112e  sw s10,48(sp)
...
1150  sw s1,24(sp)
...
1192  sw s7,60(sp)
...
11a0  sw s6,64(sp)
```

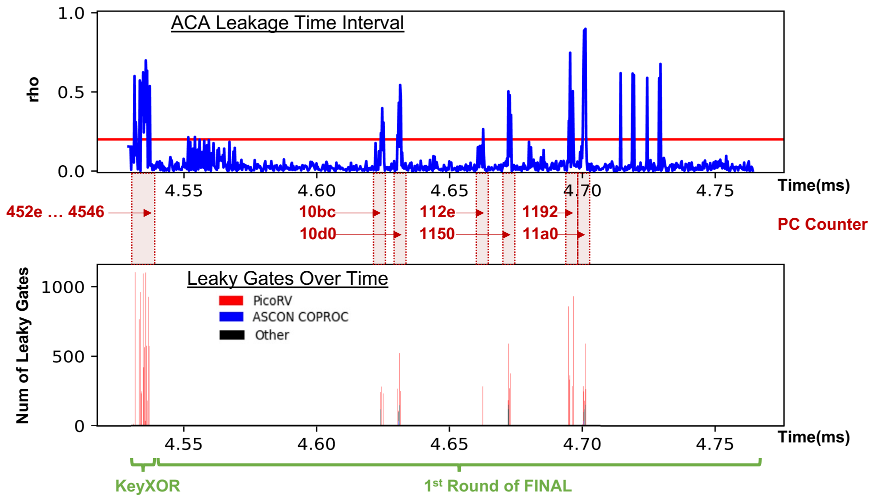

Fig. 8. ACA LTI Results (top) and Num of Leaky Gates over Simulation Time (bottom) on SWASCON-128 Key-XOR and 1st round of Finalization Phase with PC counter values in Red and ASCON Program Status in Green

SWASCON 12th round of FINAL and Key-XOR Finally, we present the root-cause analysis during the 12th round of finalization phase and Key-XOR. In this selected region, the entire simulation takes 922 clock cycles to complete. We identify the LTI with a $\rho_{threshold}$ of 0.2, which only flags 26 clock cycles out of the 922 clock cycles as containing potentially leaky samples. Figure 9 shows the root-cause analysis results from ACA. We find 14 instructions (highlighted in red in Listing 6 and Figure 9's PC Counter) that are responsible for a significant amount of leaky gates (key-dependent SCL), but still a tiny fraction of the overall 57,671 cells in the chip. We only see a few clock cycles marked as leaky at the end of finalization and this is expected. Leaky instructions include loading the final state registers values (Instr. 285a ... 28ac), loading a 128-bit key (Instr. 4558, 455a, 455c, 455e), and performing the Key-XOR operation (Instr. 4560, 4562). We see significant amount of leaky gates being flagged during the Key-XOR, and majority of the leaky gates are originated from the PicoRV.

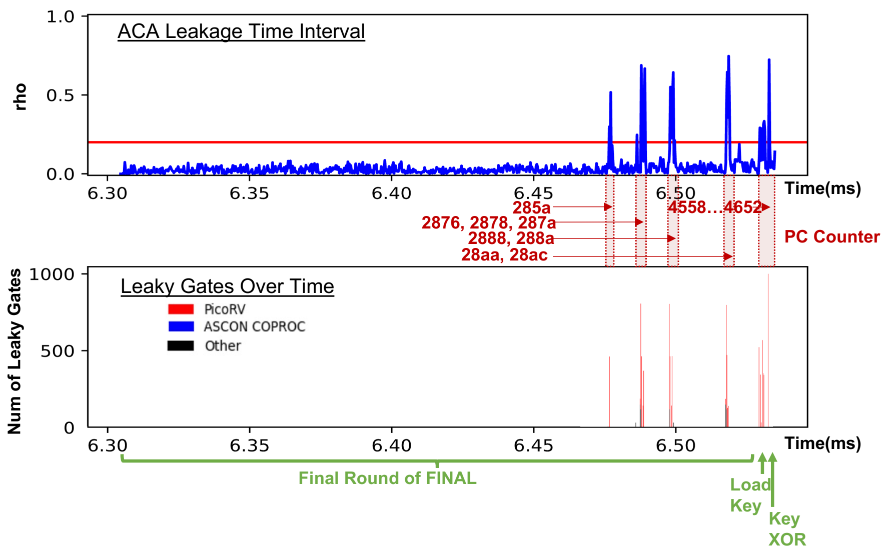

Fig. 9. ACA LTI Results (top) and Num of Leaky Gates over Simulation Time (bottom) on SWASCON-128 12th round of Finalization and Key-XOR Phase with PC counter values in Red and ASCON Program Status in Green

Listing 6. Software ASCON 12th Round of FINAL and Key-XOR. Refer to Fig. 9's X-axis

```
...
285a  lw s0,88(sp)  # load state_reg
...
2876  lw s5,68(sp)  # load state_reg
2878  lw s6,64(sp)  # load state_reg
287a  lw s7,60(sp)  # load state_reg
...
2888  lw s4,72(sp)  # load state_reg
288a  lw s3,76(sp)  # load state_reg
...
28aa  lw s10,48(sp) # load state_reg
28ac  lw s11,44(sp) # load state_reg
...
28b0: ret

4554: jal ra,fe4 <P12>
4558  lw a5,124(sp) # load key
455a  lw s5,116(sp) # load key
```

```
455c lw s2,112(sp) # load key
455e lw s1,120(sp) # load key
4560 xor s0,s0,a5  # key xor
4562 xor s5,s5,s6  # key xor
```

5 Validating Simulation against Measurement

In our final experiment, we compared the simulation results that drive the root-cause analysis with measurements taken from a prototype implementation from the same design [KLE+21]. To take the higher noise levels of the physical measurement into account, we increased the number of traces used in the measurement campaign to 50,000 (a 50x increase). The number of traces was chosen such that we identified clear positive peaks in the t-test results. The board is running at 4MHz, the sampling rate for HWASCON-128 and SWASCON-128 are 64Mhz (16 samples per clock cycle) and 16Mhz (4 samples per clock cycle).

We concluded that our t-test results for the implementation are a subset of the t-test results for the simulation. When the simulation model identifies an ACA t-test peak that corresponds to a large number of leaky gates (500 or more), then that peak is also present in the t-test on the measurements. However, the converse is not true: a t-test peak in the simulation may remain undetectable in the measurement. This happens when only very few leaky gates are involved in the t-test peak, which corresponds to minuscule power variations that remain invisible in the measurements.

Figure 10 compares the t-values for measured HWASCON-128 traces with the t-values for simulated HWASCON-128 traces. The presence of the key is clearly picked up in the simulation at KeyLoad and during the Initialization phase after KeyLoad. The t-test on the simulated traces show a similar peak pattern. Because simulated traces contain only 1000 samples, each simulated t-score graph only covers a small portion of the overall measured t-score graph.

Figure 11 compares the t-values from measured and simulated SWASCON-128 traces during the ASCON Initialization phase. The key dependence is identified in the measurements as well as in the simulation during KeyLoad, KeyXOR, first-round processing, and 12th-round processing.

Figure 12 compares the t-values from measured and simulated SWASCON-128 traces during the ASCON Finalization phase. The key dependence is identified in the measurements as well as in the simulation during KeyXOR processing befre the first-round and after the 12th-round processing.

These three examples illustrate that a meaningful connection can be made, between simulation traces and root-cause analysis (ACA) on the one hand, and between leakage patterns on simulated traces and measured traces on the other hand.

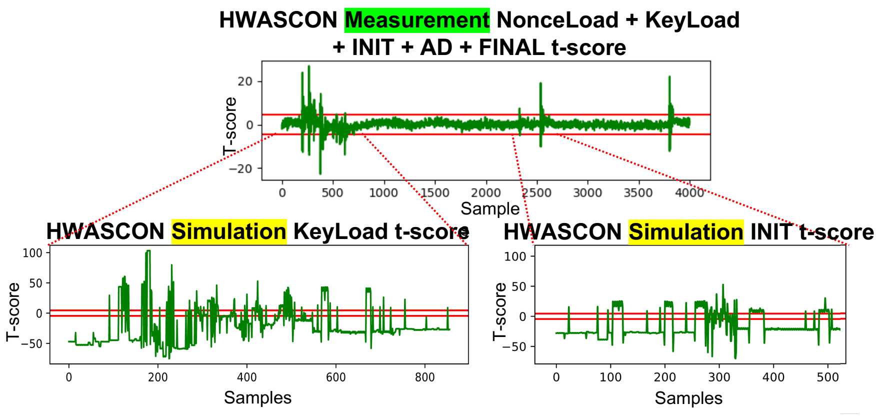

Fig. 10. Measurement and Simulation T-Test Results on HWASCON-128 KeyLoad and Initialization

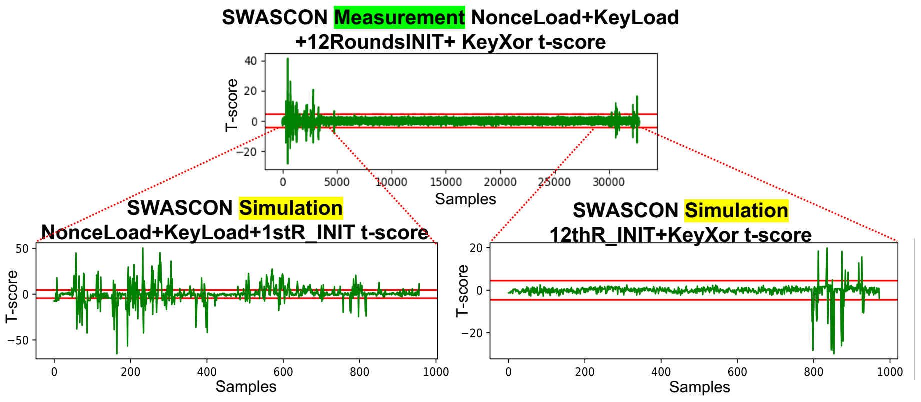

Fig. 11. Measurement and Simulation T-Test Results on SWASCON-128 Initialization

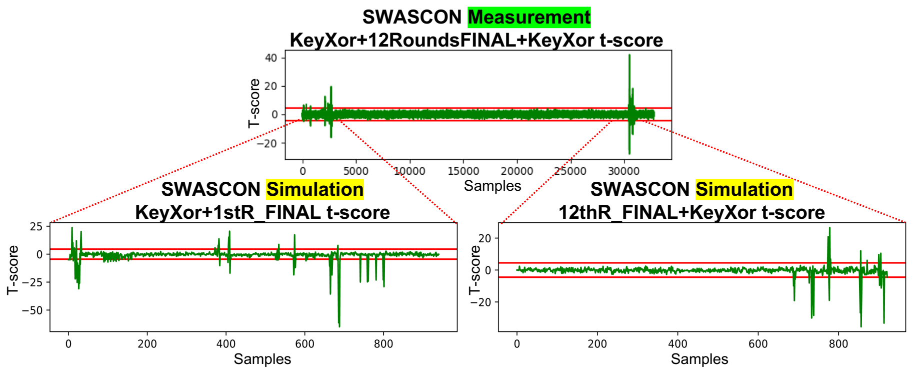

Fig. 12. Measurement and Simulation T-Test Results on SWASCON-128 Finalization

6 Conclusions

Root-Cause Analysis of Side-Channel Leakage constitutes a vital component of *Pre-Silicon* security analysis. It explains where and how the leakage comes from a implementation, thereby enabling designers to develop effective countermeasures or diagnose Side-Channel Leakage related defects. In this context, we present a comprehensive *Root-Cause Analysis* using *Architecture Correlation Analysis* on two ASCON implementations, a ASCON Coprocessor on a RISC-V SoC and a SWASCON on RISC-V (RV32IMC). These implementations have received limited attention from the research community in terms of side-channel analysis, and our study aims to address this gap by expressing the leakage pattern as a root-cause set of leaky instructions by back-annotating leaky gates and cycles to software instructions. Moreover, we validate our analysis by seeing a meaningful connection between *Pre-Silicon* simulation and *Post-Silicon* measurement traces.

References

AFM18. Alexandre Adomnicai, Jacques J. A. Fournier, and Laurent Masson. Masking the lightweight authenticated ciphers acorn and ascon in software. Cryptology ePrint Archive, Paper 2018/708, 2018. `https://eprint.iacr.org/2018/708`.

asc23. C Reference Implementation of ASCON128v12, 2023.

BBYS21. Ileana Buhan, Lejla Batina, Yuval Yarom, and Patrick Schaumont. Sok: Design tools for side-channel-aware implementations. *IACR Cryptol. ePrint Arch.*, page 497, 2021.

BIBB21. Omid Bazangani, Alexandre Iooss, Ileana Buhan, and Lejla Batina. Abby: Automating leakage modeling for side-channels analysis. Cryptology ePrint Archive, Paper 2021/1569, 2021. `https://eprint.iacr.org/2021/1569`.

DCBG$^+$17. Thomas De Cnudde, Begül Bilgin, Benedikt Gierlichs, Ventzislav Nikov, Svetla Nikova, and Vincent Rijmen. Does coupling affect the security of masked implementations? In Sylvain Guilley, editor, *Constructive Side-Channel Analysis and Secure Design*, pages 1–18, Cham, 2017. Springer International Publishing.

Fiv16. Michael Fivez. Energy efficient hardware implementations of caesar submissions. Master's thesis, ESAT COSIC, KULeuven, 2016. `https://www.esat.kuleuven.be/cosic/publications/thesis-279.pdf`.

GGB$^+$23. Ofek Gur, Tomer Gross, Davide Bellizia, François-Xavier Standaert, and Itamar Levi. An in-depth evaluation of externally amplified coupling (EAC) attacks - A concrete threat for masked cryptographic implementations. *IEEE Trans. Circuits Syst. I Regul. Pap.*, 70(2):783–796, 2023.

GS20. Ilias Giechaskiel and Jakub Szefer. Information leakage from FPGA routing and logic elements. In *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2020, San Diego, CA, USA, November 2-5, 2020*, pages 63:1–63:9. IEEE, 2020.

GWDE17. Hannes Groß, Erich Wenger, Christoph Dobraunig, and Christoph Ehrenhöfer. Ascon hardware implementations and side-channel evaluation. *Microprocess. Microsystems*, 52:470–479, 2017.

HPN$^+$19. Miao Tony He, Jungmin Park, Adib Nahiyan, Apostol Vassilev, Yier Jin, and Mark M. Tehranipoor. RTL-PSC: automated power side-channel leakage assessment at register-transfer level. In *37th IEEE VLSI Test Symposium, VTS 2019, Monterey, CA, USA, April 23-25, 2019*, pages 1–6. IEEE, 2019.

ISO16. Testing methods for the mitigation of non-invasive attack classes against cryptographic modules. Standard, International Organization for Standardization, Geneva, CH, 2016.

KLE$^+$21. Pantea Kiaei, Zhenyuan Liu, Ramazan Kaan Eren, Yuan Yao, and Patrick Schaumont. Saidoyoki: Evaluating side-channel leakage in pre- and post-silicon setting. *IACR Cryptol. ePrint Arch.*, page 1235, 2021.

KS22. Pantea Kiaei and Patrick Schaumont. Soc root canal! root cause analysis of power side-channel leakage in system-on-chip designs. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):751–773, 2022.

KYL$^+$22. Pantea Kiaei, Yuan Yao, Zhenyuan Liu, Nicole Fern, Cees-Bart Breunesse, Jasper Van Woudenberg, Kate Gillis, Alex Dich, Peter Grossmann, and Patrick Schaumont. Gate-level side-channel leakage assessment with architecture correlation analysis. *CoRR*, abs/2204.11972, 2022.

LCGD18. Yann Le Corre, Johann Großschädl, and Daniel Dinu. Micro-architectural power simulator for leakage assessment of cryptographic software on arm cortex-m3 processors. In Junfeng Fan and Benedikt Gierlichs, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 82–98, Cham, 2018. Springer International Publishing.

MBA$^+$23. Kamyar Mohajerani, Luke Beckwith, Abubakr Abdulgadir, Eduardo Ferrufino, Jens-Peter Kaps, and Kris Gaj. Sca evaluation and benchmarking of finalists in the nist lightweight cryptography standardization process. Cryptology ePrint Archive, Paper 2023/484, 2023. `https://eprint.iacr.org/2023/484`.

MMR20. Thorben Moos, Amir Moradi, and Bastian Richter. Static power side-channel analysis - an investigation of measurement factors. *IEEE Trans. Very Large Scale Integr. Syst.*, 28(2):376–389, 2020.

PGA$^+$22. Kostas Papagiannopoulos, Ognjen Glamocanin, Melissa Azouaoui, Dorian Ros, Francesco Regazzoni, and Mirjana Stojilovic. The side-channel metric cheat sheet, 2022.

PRP$^+$19. George Provelengios, Chethan Ramesh, Shivukumar B. Patil, Ken Eguro, Russell Tessier, and Daniel E. Holcomb. Characterization of long wire data leakage in deep submicron fpgas. In Kia Bazargan and Stephen Neuendorffer, editors, *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA 2019, Seaside, CA, USA, February 24-26, 2019*, pages 292–297. ACM, 2019.

RAD20. Keyvan Ramezanpour, Paul Ampadu, and William Diehl. SCARL: side-channel analysis with reinforcement learning on the ascon authenticated cipher. *CoRR*, abs/2006.03995, 2020.

SD17. Niels Samwel and Joan Daemen. DPA on hardware implementations of ascon and keyak. In *Proceedings of the Computing Frontiers Conference, CF'17, Siena, Italy, May 15-17, 2017*, pages 415–424. ACM, 2017.

SS23. Dillibabu Shanmugam and Patrick Schaumont. Improving side-channel leakage assessment using pre-silicon leakage models. In Elif Bilge Kavun and Michael Pehl, editors, *Constructive Side-Channel Analysis and Secure Design - 14th International Workshop, COSADE 2023, Munich, Germany,* *April 3-4, 2023, Proceedings*, volume 13979 of *Lecture Notes in Computer Science*, pages 105–124. Springer, 2023.

YSW+20. Yuan Yao, Patrick Schaumont, Jasper Van Woudenberg, Cees-Bart Breunesse, Edgar Mateos Santillan, and Steve Stecyk. Verification of power-based side-channel leakage through simulation. In *63rd IEEE International Midwest Symposium on Circuits and Systems, MWSCAS 2020, Springfield, MA, USA, August 9-12, 2020*, pages 1112–1115. IEEE, 2020.