Universiteit Leiden

Side-Channel Analysis of the PROACT Chip Prototype: Leakage Assessment of AES, Ascon, and Xoodyak co-processors on FPGA

Nathanael Mohanu (s3813606)

Supervisors:
T. P. Stefanov & A. Sajadi

Bachelor Thesis – Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
liacs.leidenuniv.nl

June 26, 2026

Abstract

This thesis investigates the power leakage of the PROACT chip prototype cryptographic co-processors on an FPGA board. Power traces are captured and analyzed for four different cryptography co-processors: Ascon, Xoodyak, and two AES implementations. We perform Test Vector Leakage Assessment on the four co-processors to test whether statistically significant data-dependent leakage occurs during encryption. We perform Signal-to-Noise ratio analysis to characterize leakage from the AES implementations and evaluate how leakage compares between them. By performing Correlation Power Analysis on the encryption windows, we test whether the AES cores' leakage is exploitable for recovering the cryptographic key. In this thesis we conclude that statistically significant data-dependent leakage occurs for all four co-processors, with t-values exceeding the 4.5 threshold. Furthermore, for both AES implementations, we successfully recover the full cryptography key using a set of power traces and their corresponding ciphertexts. The aim of the PROACT chip is to provide a real hardware platform to collect data for the evaluation and validation of PROACT's security simulator. By confirming that these co-processors leak power and assessing the exploitability of the leakage exhibited by the AES co-processors, we contribute empirical data to the PROACT project.

Contents

1 Introduction

A cryptographic algorithm may be considered theoretically unbreakable, yet physical attacks can exploit information that leaks from a device during computation. A device implementing the Advanced Encryption Standard (AES) algorithm may be mathematically secure against brute force algorithmic attacks, yet an attacker with physical access can recover the secret key using the leaked information and the corresponding ciphertexts. Side channel attacks exploit information leaked during cryptographic operations and provide a bridge between the theoretical security of an algorithm and the reality of its physical implementation. One side-channel attack technique is Correlation Power Analysis (CPA), introduced by Brier et al. [1] in 2004. CPA uses power-traces captured during encryption to retrieve the secret key. The exploitability of a circuit's power consumption motivates research into hardware security and countermeasures that protect algorithmic implementations. One organization that aims to aid in the development of attack-resistant devices is PROACT [2].

PROACT is a research collaboration project involving researchers from Radboud University, Leiden University, and Riscure, with the goal of aiding the development of attack-resistant hardware and algorithms, while reducing time-to-market. One of PROACT's deliverables is a hardware platform nicknamed the PROACT chip, with the goal of providing a real hardware setup for the collection of data that can be used for evaluating and validating their pre-silicon security simulator. Alongside a controller core, the PROACT chip contains a core for executing software cryptographic operations, and four hardware cryptography co-processors: Ascon, Xoodyak, and two AES implementations. This thesis aims to provide an assessment of the PROACT chip prototype cryptographic co-processors power leakage and an evaluation of the effectiveness of first-order CPA attacks on the AES implementations.

This thesis addresses the following research questions:

RQ1 Do the cryptographic co-processors of the PROACT chip prototype leak statistically significant data-dependent power consumption according to Test Vector Leakage Assessment?

RQ2 Is the observed leakage from the PROACT chip prototype AES co-processors exploitable for recovery of the secret key using first-order Correlation Power Analysis?

In this thesis, we capture power-traces from the four co-processors of the PROACT chip, prototyped on a Field Programmable Gate Array (FPGA) board. We perform Test Vector Leakage Assessment on these traces to assess whether various input variables cause data-dependent power consumption leakage. Furthermore, we use Signal-to-Noise ratio analysis to characterize AES leakage and compare leakage between the AES implementations. Lastly, we perform CPA on both AES implementations and report our findings.

As a widely used side-channel attack technique, CPA exploits the statistical relationship between power consumption and data-dependent intermediate values to recover the secret key. An AES-128 key has 2^{128} possible combinations, making a brute-force attack computationally infeasible. However, CPA can recover such a secret key in a practical time-frame using only power traces and their corresponding ciphertexts or plaintexts, making key-recovery viable in practice. CPA attacks

work by guessing an intermediate value that depends on a portion of the key, and using statistics to determine which guess portion correlates best with the traces. In this thesis, we assess the exploitability of two unprotected AES implementations' leakage on the PROACT chip prototype by performing a first-order CPA attack using power measurements and their corresponding ciphertexts. CPA offers several advantages over DPA. One such advantage that motivates its use in this thesis is that CPA allows the use of Hamming distance based models, which we elaborate on in a subsequent section. An effective target for unprotected AES implementations on FPGA boards is the state register write between rounds 9 and 10, which is modeled by the Hamming distance of their state values. A single byte from this state depends directly on the ciphertext and a single key-byte, resulting in a search-space of 256 values per key-byte. Side-channel analysis attacks on AES have been researched extensively and its exploitability is well established. In contrast, side-channel attacks on lightweight ciphers such as Ascon remain a more active research area.

Ascon [3] is a lightweight AEAD (Authenticated Encryption with Associated Data) cipher submitted to the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) in 2014 and subsequently standardized by the U.S. National Institute of Standards and Technology (NIST)[4]. Ascon uses sponge-based construction and permutations on the state. The permutations on the constructed state ensure that intermediate values depend on many different input bits, making it impossible to predict an intermediate value that depends on just one single key-byte. In a subsequent section, we briefly reflect on why CPA attacks require custom leakage models tailored to lightweight cipher implementations, drawing from related work.

Before performing a side-channel analysis attack, it is useful to assess whether any information leaks at all. To assess whether a circuit leaks data-dependent power consumption, Test Vector Leakage Assessment (TVLA) can be conducted. TVLA is a statistical method that detects leakage without requiring knowledge of how it leaks. TVLA makes a natural first step in power analysis, establishing whether leakage occurs before performing further research. In this thesis, we use TVLA to establish a baseline and assess whether the PROACT chip prototype co-processors leak data-dependent power consumption. In addition, to characterize and compare the leakage between the two AES implementations, we perform Signal-to-Noise ratio (SNR) analysis, quantifying leakage attributed to intermediate values used by various leakage models.

By performing TVLA analysis on the cryptographic co-processors and CPA on the AES co-processors, we assess the extend to which the PROACT prototype leaks power and whether the AES leakage is exploitable for key-recovery. Furthermore, using Signal-to-Noise ratio analysis, we compare leakage between the AES implementations. With the results presented in this thesis, we contribute empirical data to the PROACT project. Furthermore, we provide the used firmware and a set of custom built scripts that contain pipelines for trace collection, SNR analysis, TVLA analysis, and CPA. The analysis scripts take the trace sets as input and produce a summary of statistics and plots where applicable.

1.1 Thesis structure

First, we provide the background information required for interpreting this thesis in Section 2, covering the target algorithms, side-channel analysis, the hardware and software platforms, and the analysis strategies used in this thesis. In Section 3, we discuss various related papers and explain

how they are linked to this thesis. In Section 4, we explain our methodology and experimental setup, covering our capturing setup, the captured power-trace sets, and briefly explain the scripts used for analysis. In Section 5, we present and discuss the TVLA results. In Section 6, we present and discuss the results of our SNR analysis and CPA attacks. In Section 7, we give an overview of our conclusions and the limitations that should be considered. Finally, in Section 8, we discuss several opportunities for future research.

2 Background

To correctly interpret the results discussed in this thesis, some background knowledge is recommended. First, properties of the target algorithms that are relevant to this thesis are provided in Section 2.1. Secondly, side-channel attacks and power analysis are introduced in Section 2.2. In Section 2.3, we cover ChipWhisperer, from which this thesis uses both hardware and software. In Section 2.4, we give an overview and explanation of the leakage models that are used in this thesis. Then, Test Vector Leakage Assessment and Signal-to-Noise Ratio analysis are introduced in Section 2.5. Finally, we give a brief overview of the PROACT hardware in Section 2.6.

2.1 Target Algorithms

The PROACT chip deploys four co-processors, each being a hardware implementation of a cryptographic algorithm. First, an overview is given of the AES algorithm in Section 2.1.1. We discuss the two implementations of AES on the PROACT chip and outline their differences in Section 2.1.2. A general overview is given about AEAD algorithms and their differences to AES in Section 2.1.3. Finally, we discuss Ascon and Xoodyak, which are both AEAD algorithms, in Sections 2.1.4 and 2.1.5, respectively.

2.1.1 Advanced Encryption Standard

Advanced Encryption Standard (AES) is a cryptographic algorithm designed by Joan Daemen and Vincent Rijmen [5] and was standardized by NIST [6] in 2001 as the successor to Data Encryption Standard (DES).

AES is a block cipher that uses a key of 128, 192, or 256 bits; this thesis focuses on AES-128 specifically. The AES algorithm divides the input data into 16-byte blocks, where a data block at any given point is also referred to as the state. A 16-byte data block can be viewed as a 4×4 matrix with one byte in each cell. During encryption, the matrix goes through a series of byte and matrix operations. The AES algorithm performs a number of rounds of these operations depending on the AES implementation. AES-128 performs 10 rounds. Before the first round begins, the secret key is XOR-ed into the state, also called key-whitening.

AES applies four different transformation layers, performing operations on the bytes or state matrix. In each AES round, all four layers are applied in a fixed order: SubBytes, ShiftRows, MixColumns, AddRoundKey. The MixColumns operation is skipped in the last round. A diagram of the AES-128 encryption flow is given in Figure 1.

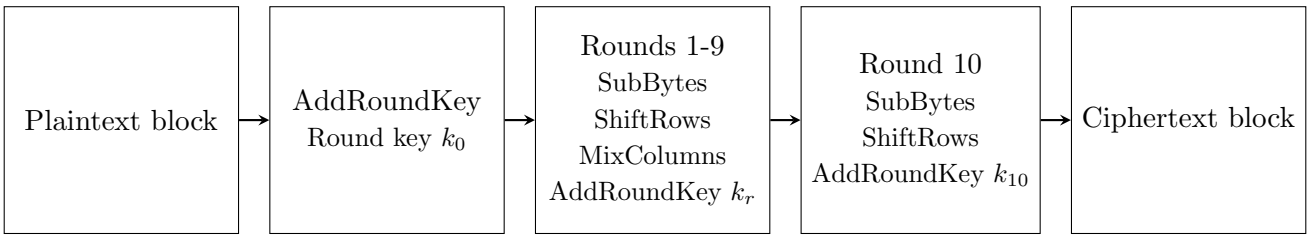

Figure 1: Diagram of the cryptographic flow for a 10-round AES encryption on an input data block.

In each AES encryption round, several transformation layers are applied to the 16-byte data block:

1. SubBytes: each byte is replaced by another byte using a bijection mapping. For this operation, a lookup-table-based substitution box (S-box) is commonly used. A diagram for the SubBytes layer is given in Figure 2.

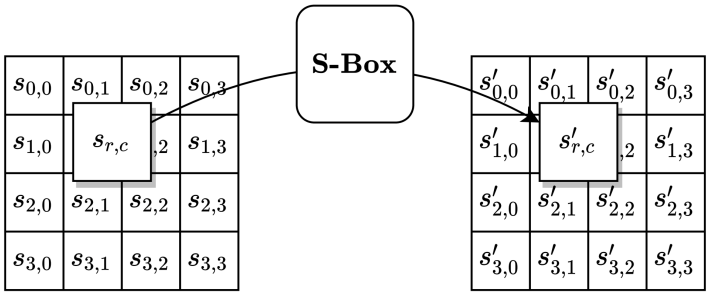

Figure 2: Diagram of the sub bytes layer of the AES algorithm (figure taken from [6]).

2. ShiftRows: performs shifts on the grid as displayed in Figure 3:
   (a) The second row is shifted to the left by one position.
   (b) The third row is shifted to the left by two positions.
   (c) The fourth row is shifted to the left by three position.

   Note that the first row remains untouched by the ShiftRows layer.

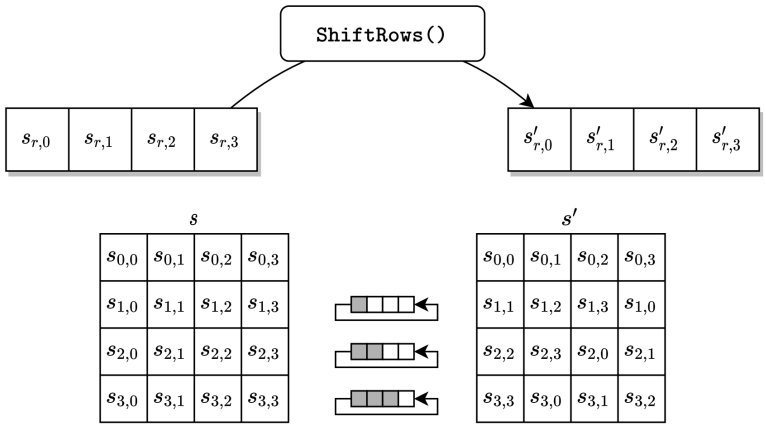

Figure 3: Diagram of the ShiftRows layer of AES (figure from [6]).

3. MixColumns: a matrix multiplication is performed on each state column as displayed in Figure 4. The MixColumns layer applies diffusion to the algorithm, causing output bytes to depend on multiple input bytes.

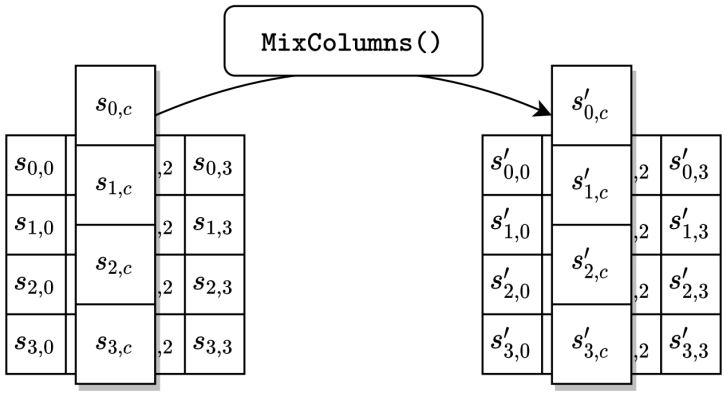

Figure 4: Diagram of the AES Column matrix multiplication (figure taken from [6]).

4. AddRoundKey: the state is XOR-ed with the round-key k_r. k_r is determined by the AES *KeyExpansion* algorithm. This algorithm transforms the original 128-bit key into $Nr + 1$ round keys, where Nr denotes the amount of total rounds that are to be performed on the data block [1]. k_0 is applied before the start of the first round during the key-whitening operation. The algorithm is fully deterministic and reversible, i.e., any k_r can be transformed back into k_0, making the last round's key a practical target as it is used to produce the final ciphertext.

Each round is performed on a single 16-byte block and the AES mode determines how multiple data-blocks are chained together. One example of an AES mode is the CBC mode, which first XORs each data-block with the ciphertext corresponding to the previous block. As the PROACT

[1] For AES-128, a total of 11 keys are generated, where k_0 equals the original secret key.

chip's AES implementations operate on a single 16-byte block, we do not explain modes in this thesis.

For CPA, a commonly used target is the last AES encryption round. In the last round, MixColumns is skipped, making each ciphertext byte map directly to a byte from round 9's output state. When hypothesizing a key-byte with index b, we can reverse ciphertext_b into the output state byte of round 9 and use the Hamming distance of the two states and use the result to predict power consumption. In Section 2.4, we explain this specific leakage model in further depth.

2.1.2 PROACT - AES1 and AES2 comparison

The PROACT chip includes two AES co-processors. These are referred to as AES1 and AES2. Both AES1 and AES2 implement AES-128 but implement the SubBytes layer differently. The difference is that the AES1 co-processor uses a look-up table for the substitution box, while AES2 uses a logic-based bit-wise substitution box implementation.

As the implementations differ on an architectural level, they may produce different power profiles, which we assess in Section 6.1.

2.1.3 AEAD Algorithms

Authenticated Encryption with Associated Data (AEAD) algorithms combine encryption with authentication. Unlike AES, which has two inputs (the key and a plaintext) and one output (the ciphertext), AEAD algorithms typically take four inputs and produce two outputs [2]. Besides the plaintext and cryptographic key, AEAD algorithms take a nonce and associated data as additional inputs and produce a tag as additional output. The nonce is an additional unique value that ensures the uniqueness of an individual encryption process. The associated data is authenticated alongside the plaintext but is not encrypted, allowing the receiver to recompute the tag and verify it against the received value. This process ensures the integrity and authenticity of both the ciphertext and the associated data.

2.1.4 Ascon

Ascon [3] is one representative of AEAD algorithms. Ascon was designed in 2014 by Dobraunig, Eichlseder, Mendel, and Schläffer. In 2019, Ascon was selected as the first choice lightweight encryption recommendation in the CAESAR competition [7]. In 2023, Ascon was selected as the NIST lightweight cryptography standard [4]. Multiple variants of the algorithm exist but this thesis focuses specifically on Ascon-128, therefore when we refer to Ascon we refer to Ascon-128. An overview of the Ascon encryption flow is given in Figure 5.

[2]The tag and ciphertext may be concatenated together depending on the implementation.

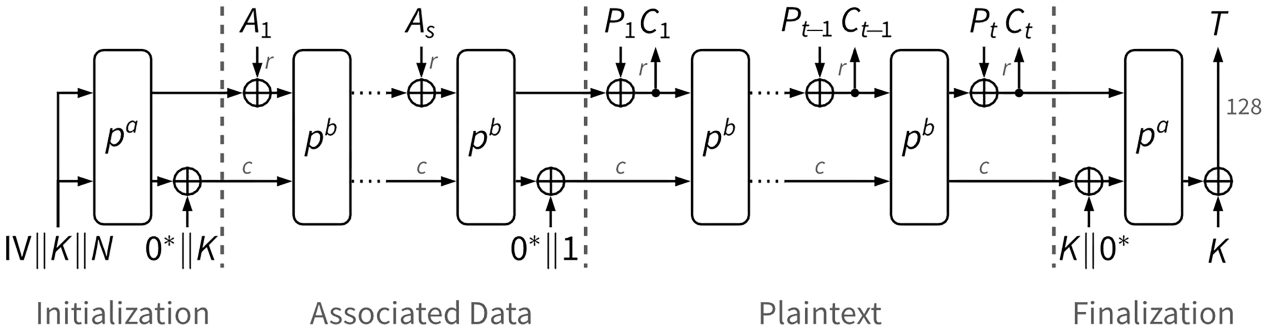

Figure 5: Diagram of the Ascon encryption flow (figure taken from [8]).

Ascon performs a rounds of permutation function p on a 320-bit initial state that consists of 5 concatenated 64-bit words. This initial state consists of IV, the secret key, and the nonce, where IV is an Ascon constant that specifies the algorithm and parameters such as the key size. After the initialization phase, the key is XOR-ed into the state. In the second phase, the associated data A is absorbed in blocks, followed by b permutations per block. Before the plaintext phase starts, a 1-bit domain separation constant is XOR-ed into the state. The plaintext is XOR-ed into the state one block at a time, followed by the extraction of a ciphertext block and b permutations. In the finalization phase, the key is XOR-ed into the state again, after which the tag T is produced by performing another a rounds of p and a final additional key XOR into the state. For Ascon-128, $a = 12$ and $b = 6$.

In each permutation round of p, the following three transformation layers are applied:

1. Constant addition layer: the corresponding round constant c_i is XOR-ed into the state.
   Formally, $c_i = \text{const}_{16-\text{rnd}+i}$ where i is the permutation round and **rnd** is the total number of rounds for the current phase [9].

   The round constants are byte values where the most significant half of the byte counts down from 0xf, and the least significant half counts up from 0, allowing for easy computation of the constant.

2. Substitution (S-box) layer: substitution layer using a 5-bit S-box.
   Formally, $(s_{(0,j)}, s_{(1,j)}, ..., s_{(4,j)}) = \text{SBox}(s_{(0,j)}, s_{(1,j)}, ..., s_{(4,j)}$ [9]

3. Linear diffusion layer: each of the 64-bit words is mixed with two rotated versions of itself using XOR operations.
   Formally, the layer applies linear function Σ_i to state word S_i. For $0 \leq i \leq 4$. Where Σ_i is

defined as:

$$\Sigma_0(x_0) = x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28) \quad (1)$$
$$\Sigma_1(x_1) = x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39) \quad (2)$$
$$\Sigma_2(x_2) = x_2 \oplus (x_2 \ggg 1) \oplus (x_2 \ggg 6) \quad (3)$$
$$\Sigma_3(x_3) = x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17) \quad (4)$$
$$\Sigma_4(x_4) = x_4 \oplus (x_4 \ggg 7) \oplus (x_4 \ggg 41) \quad (5)$$

In this thesis, the most important property of Ascon, that is considered, is the key absorption into the state and the permutation rounds on this state. After the initial permutation rounds, each bit of the state depends on many different input bits.

2.1.5 Xoodyak

Similar to Ascon, Xoodyak is an AEAD algorithm. Xoodyak is designed by Daemen, Hoffert, Mella, Peeters, Van Assche, and Van Keer [10] and was a NIST lightweight cryptography finalist but was not selected as the standard in 2023. A brief overview of the algorithm is given here. For an in-depth specification of Xoodyak we refer the reader to [11].
Like Ascon, Xoodyak uses sponge-based state absorption but applies a different permutation function and operates on a 384-bit state. The 384-bit state consists of three *planes* of 128 bits each. These 128-bit planes are arranged as four 32-bit lanes. Xoodyak applies 12 rounds. Each round consists of the following five steps [11]:

1. A mixing layer θ: mixes each column with its neighbors, providing diffusion across the state.
2. Plane shifting ρ_{west}: performs rotations on the planes.
3. Round constant addition ι: XOR c_i into the state.
4. A nonlinear layer χ: applies XOR and AND operations on each 3-bit column, providing confusion [3].
5. Plane shifting ρ_{east}: performs rotations on the planes.

Xoodyak uses the Cyclist mode of operation [11]. Cyclist alternates between two types of operations: absorbing input data into the state, and outputting data from the state by reading bytes from the permutation output. Similar to Ascon, Xoodyak takes a nonce and associated data and produces a tag along with the ciphertext for authenticated encryption.

In this thesis, similar to Ascon, the most important property of Xoodyak, that is considered, is the sponge-based construction and the state permutations.

2.2 Side-Channel Analysis

Side-channel Analysis (SCA) exploits information that is leaked during computations with the aim to recover secret information, usually being the cryptographic key. Different properties of a hardware

[3]The confusion property obscures the relationship between the key and ciphertext.

device like sound and electromagnetic waves can be used to exploit information leakage, but for the purpose of this thesis, SCA will be explained in the context of exploiting power consumption as a source of leakage.

2.2.1 Power Analysis

Complementary Metal-Oxide Semiconductor (CMOS) circuits are the foundation of modern electronics. Relevant to power analysis is that the dynamic power consumed by a CMOS circuit is proportional to the occurring transistor switching activity [12]. Therefore, the dynamic power consumption is influenced by the data being processed. Because dynamic power consumption is secondarily influenced by other factors like noise, other operations, and even external factors, many power traces are usually needed to cancel out these factors and derive meaningful correlations. Power analysis attacks exploit these leaking correlations in an attempt to recover secret information. Two widely used strategies are Differential Power Analysis (DPA) [13] and Correlation Power Analysis (CPA) [1]. This thesis uses CPA as the power analysis strategy. We give an example of a plotted power trace in Figure 6 to provide a visual representation of a power trace to the reader.

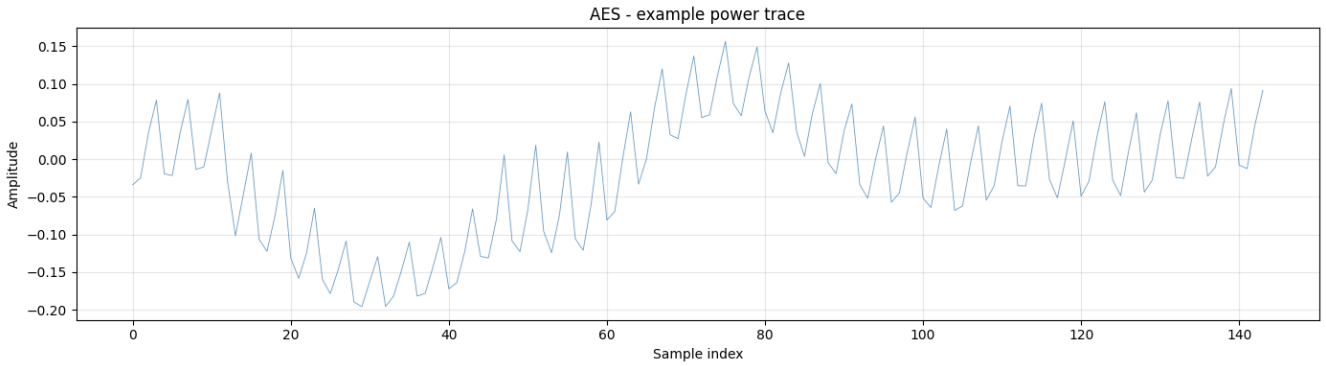

Figure 6: An example of a captured power-trace during AES encryption. Full cryptography window - 144 samples (36 clock cycles).

2.2.2 Correlation Power Analysis

Correlation Power Analysis, also referred to as CPA, was first introduced in 2004 by Brier et al. [1] as an improvement over Differential Power Analysis (DPA). In DPA, a key-byte is hypothesized and traces are divided into two groups based on a specific bit of the predicted intermediate value. If the key-byte guess matches the real key, the two groups will have systematically different power consumption, causing a spike in the difference of means. If the guess was wrong, the difference of means is essentially flat. The correct key-byte guess is the guess producing the largest spike.

In CPA, the attacker uses a leakage model to predict the power consumption and computes the correlation between this prediction and the actual power traces. The key guess that results in the highest correlation between the prediction and real power samples is concluded to be the correct guess. A commonly used leakage model is the Hamming Distance. The Hamming Distance is defined as:

$$\mathrm{HD}(d_1, d_2) = \mathrm{HW}(d_1 \oplus d_2) \qquad \text{(Hamming Distance)}$$

where the Hamming weight HW(x) is the number of 1-bits in the binary string x, and d_1 and d_2 are two values for which the Hamming distance is computed[4]. The Hamming weight can also be used to model the power consumption. However, as we discuss in subsequent sections, the Hamming Distance better fits our scenario.

By using the Hamming distance as the power prediction model, the attacker calculates the amount of toggled bits between consecutive states and therefore approximates the transistor switching activity occurring on the circuit. For the attack procedure, the attacker hypothesizes a value for the targeted portion of the key and computes the predicted power consumption. CPA uses the Pearson correlation between the predicted power values for each key-byte guess and the captured real power traces:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}} \qquad \text{(Pearson Correlation)}$$

where i is a power trace, x_i is the predicted value for that trace, y_i is the measured power consumption of trace i at a given sample point, and $\bar{x}$, $\bar{y}$ are the means of the predicted values and measured power values, respectively. This correlation is calculated at every sample point, resulting in a correlation array. The Pearson Correlation tells us the correlation strength between the predicted power consumption and the actual data at each sample point. The key-byte with the highest Pearson Correlation is concluded to be the correct key guess. For the Pearson Correlation, the area of interest does not need to be isolated. However, isolating a leakage window can reduce noise and computational demand.

The leakage model uses a selection function to target an intermediate value that depends on a known value and a portion of the key. In the case of AES, the attacker typically targets an intermediate value that depends on a single byte of the key and one byte of the ciphertext or plaintext, depending on the targeted operations.

In the case of AES-128, isolating each key byte and performing CPA reduces the search space from 2^{128} to 16 independent searches of 2^8 possible values, making a CPA attack computationally feasible.

2.3 ChipWhisperer

ChipWhisperer [14], an open-source tool-chain maintained by NewAE Technology [15], provides hardware and software tools focused on side-channel analysis. In this thesis, we use both hardware and software provided by ChipWhisperer. The hardware is used for trace capturing, the software is used for performing the CPA attacks.

2.3.1 ChipWhisperer Hardware

For capturing the power traces, two ChipWhisperer devices are used in our setup: a target board, and a capturing device. The target device used in this thesis is the CW305 [16], which utilizes the Artix-7 XC7A100T FPGA. The PROACT chip prototype is equipped with a general RISC-V processor that can be programmed with custom firmware. This processor will be used to communicate with the host (in our case a PC) over USB using a UART-based communication

[4]In this thesis, d_1 and d_2 are state values corresponding to the targeted AES rounds.

channel and to communicate to the PROACT chip prototype co-processors, which are also deployed on the FPGA. The CW305 implements a PLL that can be programmed over USB and supports frequencies from roughly 5MHz to 160MHz [16].

For capturing the power traces, the ChipWhisperer-Husky device is used [17]. The Husky is a capturing device that is connected to the target using a 20-pin cable. This connection can be used to capture the trigger and the clock signal from the target. Furthermore, a coax cable is connected between the Husky and the CW305 which is used to record the power consumption. The Husky supports frequencies up to 200 MHz, which allows us to capture four samples per clock cycle when running the target at 50 MHz.

2.3.2 ChipWhisperer Software

ChipWhisperer provides open source software [18] that can be used to capture traces and perform CPA. Within the software package, various APIs are provided that we use in this thesis.

First, using the Scope API [19], we are able to connect the Python program to our ChipWhisperer Husky. Using this API, we are able to program various settings onto the Husky and command it from the host. Using the target API [20], we connect to the target board, allowing us to program it and load the PROACT bit-stream onto the FPGA. The capture API [21] contains the functions that allow the creation of projects, and storing the captured traces with corresponding data into them.

In the Analyzer API [22], a set of default AES specific leakage models are included. The leakage models provided by ChipWhisperer that we use are discussed in Section 2.4. The API provides a framework, allowing us to specify a leakage model and run an attack on a project that contains the traces and their corresponding data. Furthermore, the flexibility of the API allows users to create custom leakage models and use them for an attack. The attack results contain several statistics, such as the best key guess, correlations, and true key byte ranks.

2.4 CPA Leakage Models

In this section, we list and explain the leakage models we use in this thesis. As we explain in Section 3, Hamming distance based models best fit our scenario and thus only these models are used in this thesis. Furthermore, we only perform CPA using the non-pipe-lined models, as our traces contain data from a single 16-byte AES encryption. We perform CPA on our AES traces using the following leakage models provided by the ChipWhisperer API:

1. `last_round_state_diff`: Hamming distance of the AES state between the 9th and 10th round. This model depends on knowing the ciphertext. A diagram of this leakage model is presented in Figure 7. s_b^9 and s_b^{10} are defined as:
   (a) s_b^9: The state after round 9, before round 10. Computed by reversing the AddRoundKey operation using the guessed key-byte with index b, and reversing the SubBytes operation. Note that MixColumns is skipped in this round and does not have to be reversed.
   (b) s_b^{10}: The ciphertext byte whose position is corrected for round 10's ShiftRows operation.

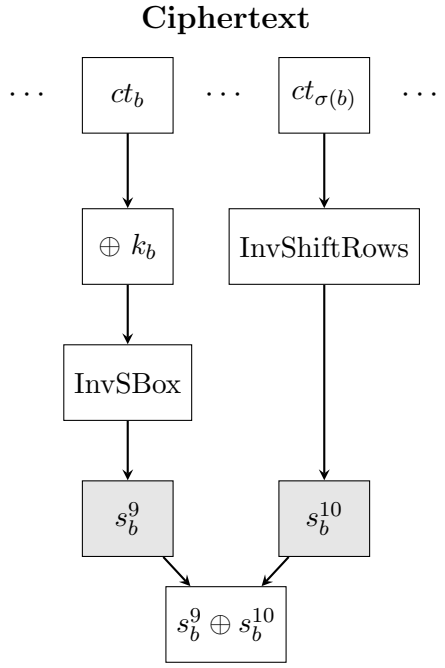

Figure 7: Diagram of the `last_round_state_diff` leakage model. Where σ corrects some byte position b for the ShiftRows operation.

By taking the Hamming Distance of s_b^9 and s_b^{10}, we model the power consumption based on the switching activity that occurred during the register write between these states.

The produced key guess corresponds to the last round key. Using ChipWhisperer's `key_schedule_rounds` function, we are able to transform k_{10} back into k_0.

2. `last_round_state_diff_alternate`: similar to `last_round_state_diff` but does not account s_b^{10} for round 10's ShiftRows operation.
3. `sbox_in_out_diff`: computes the Hamming Distance between the input and output of the SubBytes layer in the first round. Depends on knowledge of the plaintext.
4. `round_1_2_state_diff_text`: Hamming distance between the AES plaintext and the mix columns output. Depends on knowledge of the plaintext.
5. `round_1_2_state_diff_key_mix`: similar to `round_1_2_state_diff_text` but applies the initial k_0 XOR on p_b as well.
6. `round_1_2_state_diff_sbox`: similar to `round_1_2_state_diff_key_mix` but applies SubBytes to s_0 as well.

Note that leakage models 3-6 all depend on intermediary register writes during rounds. Models 1 and 2 depend on a state register write between the final two rounds.

Furthermore, we use one custom leakage model in this thesis. This leakage model calculates

the Hamming distance between the full outputs of round 1 and 2. To keep consistent with ChipWhisperer's naming convention, we call this model `custom_1_2_state_diff`.

2.5 Analysis Methods

This thesis uses two methods for detecting and evaluating leakage on the target platform. First, Test Vector Leakage Assessment is used to establish a baseline and to determine whether any leakage occurs. Second, Signal-to-Noise ratios are computed and plotted to characterize and compare leakage between the two AES implementations.

2.5.1 Test Vector Leakage Assessment

Test Vector Leakage Assessment (TVLA) is a method introduced in 2011 by Goodwill et al. [23] and is used to detect data-dependent power leakage. Unlike CPA, which requires a power leakage model and intermediate values to correlate against, TVLA requires neither. TVLA tests whether data-dependent power consumption occurs at all, without requiring knowledge of how. Because TVLA requires no power leakage model, TVLA is a natural first step for assessing whether leakage occurs and establishing a baseline.

TVLA works by comparing two sets of power traces, often named group A and group B, and calculating whether their difference is statistically significant. The null hypothesis (H_0) states that the groups are not statistically different, the alternative hypothesis (H_a) states that the groups have a statistically significant difference. The statistical significance of this difference is evaluated by performing Welch's t-test:

$$t = \frac{\bar{X}_A - \bar{X}_B}{\sqrt{\frac{s_A^2}{N_A}\frac{s_B^2}{N_B}}} \qquad \text{(Welch's t-test)}$$

where $\bar{X}_A$, $\bar{X}_B$ are the means of the compared groups, s_A^2, s_B^2 are the sample variances of the groups, and N_A, N_B are the sizes of the groups. The resulting t-value indicates the level of statistical significance. As proposed by Goodwill et al. [23], a t-value exceeding the threshold of 4.5 indicates a statistically significant difference between the sets, implying power consumption leakage.

The leakage that occurs depends on the context of the two groups. By using a fixed group A, and a group B that has a varying parameter between traces, any statistically significant difference between these groups can be attributed to this varying parameter.

Formally, let E be a cryptographic function with parameter set P. We define:

1. **Group A:** A set of traces where a trace $T \in A$ is a vector of power measurements obtained by executing cryptographic function E, where each parameter $p \in P$ of E is fixed.
2. **Group B:** A set of traces where a trace $T \in B$ is a vector of power measurements obtained by executing the encryption function E, where each parameter $p \in P$ of E is fixed except for a one parameter p', which has a value that varies randomly between traces in B.

Because only one parameter varies randomly, any statistically significant difference between the groups can be attributed to p'. While TVLA can confirm leakage, it does not imply, however, that the detected leakage can be exploited for key recovery.

In this thesis, we apply TVLA to 8 pairs of trace sets to test whether data-dependent leakage is present. We test p-dependent power leakage for each $p \in P$ except the key parameter p_k.

2.5.2 Signal-to-Noise Ratio Analysis

Signal-to-Noise Ratio analysis (SNR analysis) is a test that can be used to characterize power leakage. Mangard et al. [24] demonstrate how SNR analysis can be used to quantify leakage attributed to selected intermediate values. The distinguishing property of SNR analysis when compared to TVLA is that SNR analysis requires a known intermediate value. However, as a result the user can characterize the leakage by quantifying the variance explained by this intermediate value.

SNR analysis is performed by grouping traces based on intermediate values that were encountered during capturing. This means that to perform SNR analysis using intermediate values, the intermediate value z corresponding to a trace T must be either known or reproducible.

SNR groups traces by their encountered value of z, resulting in $|Z|$ groups, where $|Z|$ is the number of distinctly observed values of z. For $|T|$ traces, a group contains $|T|/|Z|$ traces on average. In this thesis, we perform SNR analysis using a single byte as our intermediate value. Performing SNR on a single byte using 10000 traces results in an approximate minimum of $10000/256 \approx 39$ traces per group [5].

After dividing the traces into groups, between-group signal variance and noise variance are calculated. The signal variance tells us how much the group means vary across intermediate values. Noise variance is the average in-group variance across intermediate values. The formula for the SNR value is defined as:

$$\text{SNR} = \frac{\text{Var}(\mathbb{E}[L|Z])}{\mathbb{E}[\text{Var}(L|Z)]} \qquad \text{(The SNR formula)}$$

where L is the leakage sample and Z is the intermediate.

An example of an SNR plot is provided in Figure 8. The figure displays the SNR values of a set of power traces, obtained by executing the AES algorithm, using the S-box outputs as the intermediate values.

[5] If less than 256 distinct values were encountered for the intermediate, the approximated minimum group size is larger.

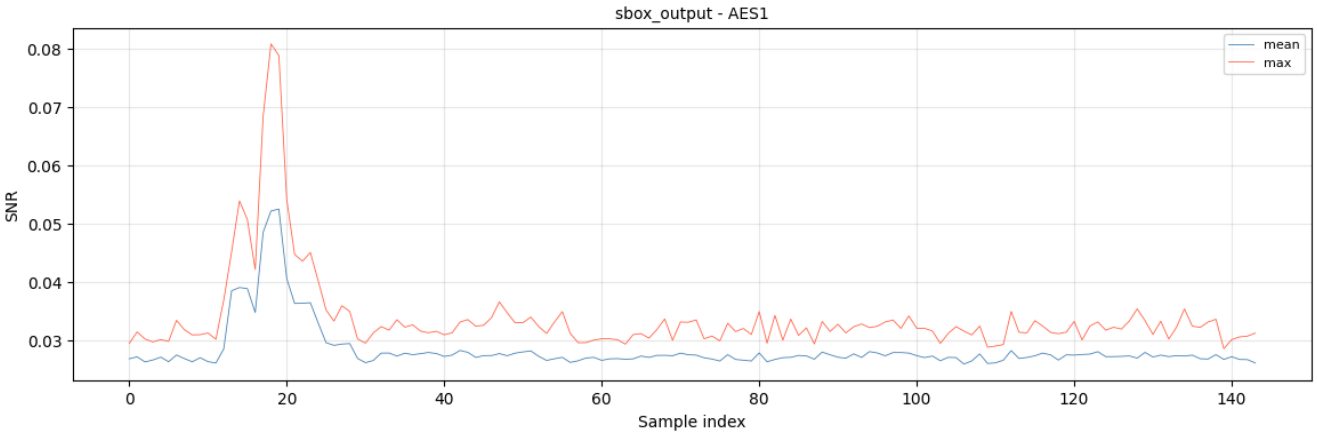

Figure 8: An SNR plot on AES traces - Hamming Distance of the sbox input and output as intermediate values - mean and max across all 16 bytes.

One noteworthy caveat for SNR analysis is that SNR quantifies leakage magnitude [24] but unlike TVLA, has no universal decision criterion. Furthermore, an increase of SNR does not imply that the sample points are exploitable. Consequently, in this thesis, SNR is used as a complementary, descriptive tool to characterize leakage and compare leakage between the AES co-processors. We do not use the SNR results to guide our CPA attack in Section 6.1. However, in Section 6.3, we perform an additional CPA attack using a selection of leakage models and SNR guided leakage windows to validate whether the observed SNR peaks align with the sample points exploited by the CPA attack in Section 6.1.

2.6 The PROACT chip Hardware

To give the reader a better understanding of the target hardware, we give a brief overview of the PROACT chip hardware. In this thesis, we target the prototype of this hardware design on an FPGA board.

The hardware design of the PROACT chip prototype consists of two IBEX cores and four hardware co-processors. The co-processors implement Ascon, Xoodyak, and two AES implementations. The IBEX core is an open-source 32-bit RISC-V CPU written in SystemVerilog. A diagram of the IBEX core is given in Figure 9. One of the IBEX cores functions as a controller and can be programmed with custom firmware over the Serial Peripheral Interface (SPI) protocol. This controller establishes a UART-based communication channel that can be used to communicate with other devices. Furthermore, the controller can be utilized to write data to the other IBEX core and the cryptographic co-processors. The other IBEX core is used to execute software cryptography instructions but is not used in this thesis.

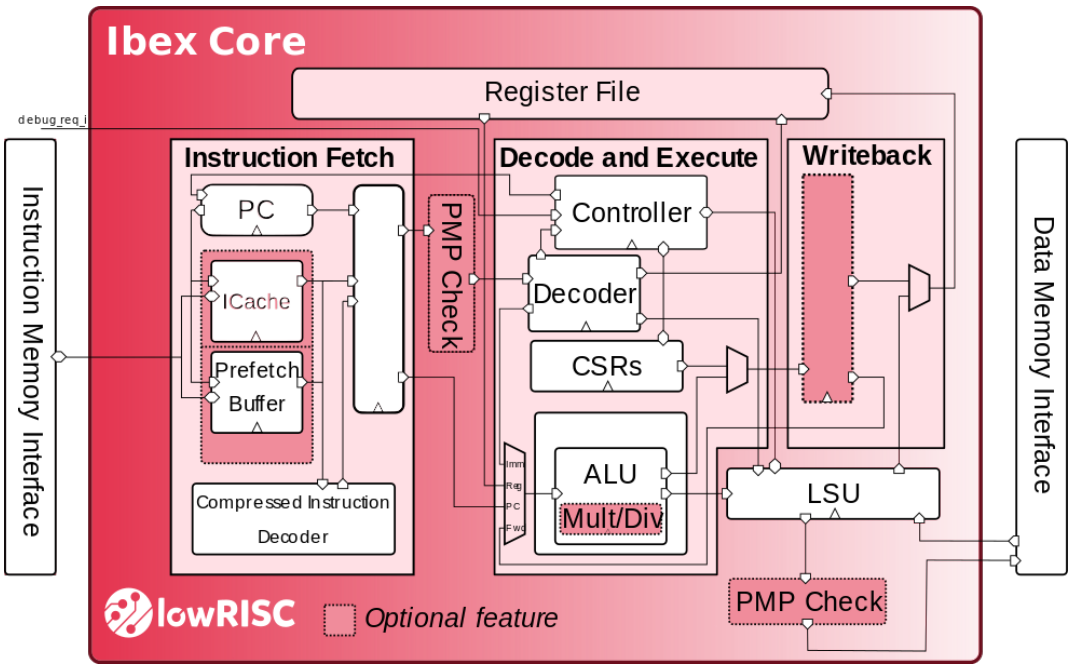

Figure 9: Diagram of the Ibex Core.

Internally, the hardware is designed with two 32-bit registers for management purposes: a status register and a control register. The status register contains bits reflecting the state of the hardware components and is used to read their current status. The controller register is managed by the controller and is used to control the other co-processors. A block diagram of the PROACT hardware, excluding the four co-processors, is displayed in Figure 10.

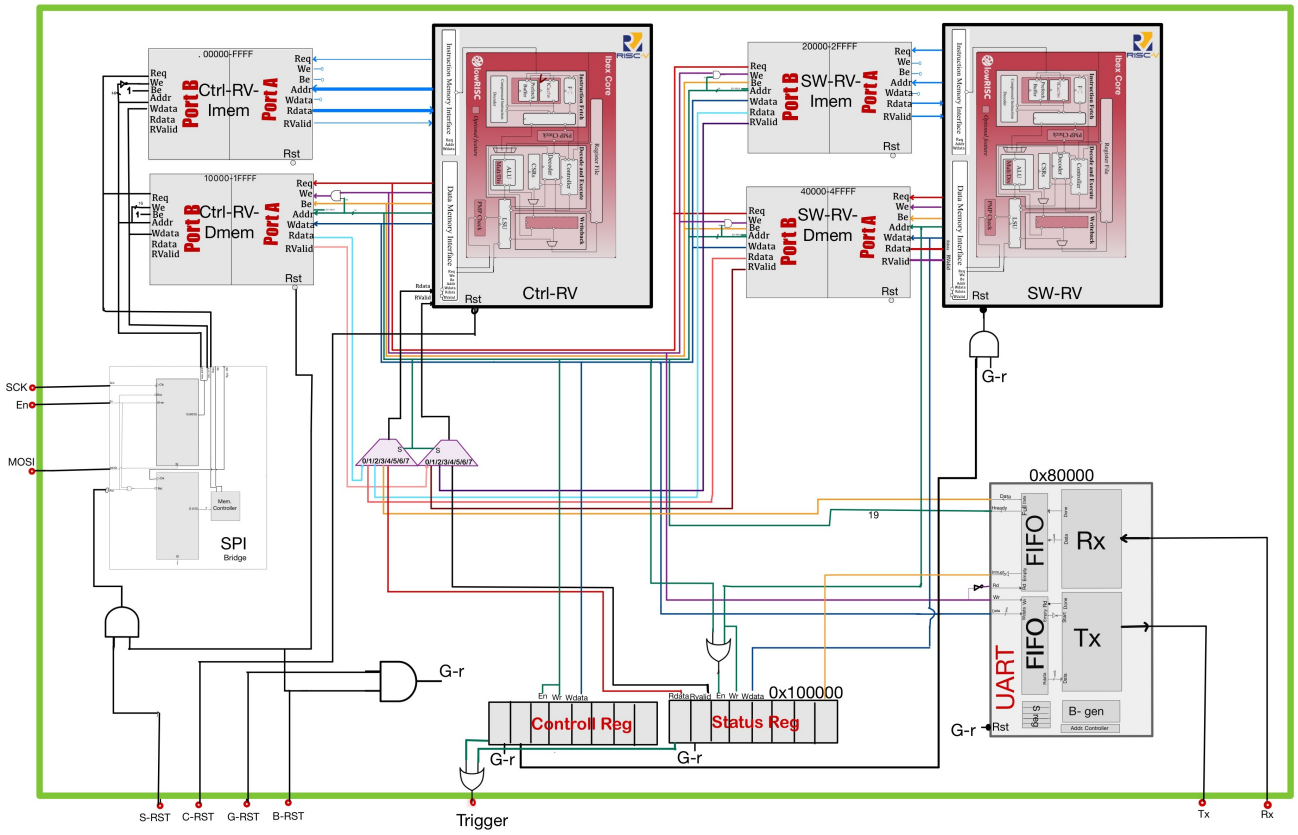

Figure 10: Diagram of the PROACT hardware, excluding the cryptographic co-processors.

Besides the co-processors, the most important properties of the hardware, that are considered for this thesis, are the IBEX controller core, the two registers used by the controller, and the UART-based communication channel. We will elaborate on this setup in further detail in Section 4.1.

3 Related Work

TVLA [23] has become a widely adopted method for identifying power leakage from power traces. A 2023 paper by Wang and Tang [25] provides an extensive assessment on the strengths and weaknesses of TVLA. In this paper, the authors mention several drawbacks of TVLA. One of the disadvantages discussed is the unreliability of positive outcomes. They explain that tests on long power traces are more likely to result in at least one false positive. Moreover, it is often difficult to reason about negative outcomes. The authors explain how the test can be used to either reject the null hypothesis or fail to reject it, but the test cannot be used to prove the null hypothesis. The authors also state that simple limitations like equipment quality can affect the fairness of the test, resulting in negative outcomes being challenging to explain. Furthermore, the authors note that TVLA can only confirm whether leakage exists, but the test provides no information regarding the specific location of the leakage or how to exploit it [25]. In this thesis, TVLA is used to establish a baseline, confirming whether statistically significant leakage exists. As a positive result does not

imply exploitability, separate research is required to confirm whether the leakage can be exploited. We perform CPA in addition to TVLA in an attempt to confirm exploitability of the leakage for the AES implementations. Whether leakage from the Ascon and Xoodyak implementations is exploitable remains open for future research.

For TVLA analysis, Goodwill et al. [23] recommend capturing traces in an interleaved manner, meaning that traces from the fixed and varying sets are captured in a random order. To prevent systemic bias due to factors such as temperature change on the equipment, we capture the traces in an interleaved manner.

Mangard et al. [24] described how Signal-to-Noise ratio analysis can be used to quantify data-dependent leakage. The main difference between TVLA and SNR Analysis is that TVLA detects data-dependent leakage using Welch's t-test, whereas SNR Analysis measures the ratio of data-dependent variance to noise variance using an intermediate value. Because TVLA only tells us whether a side-channel leak occurs, SNR can be a useful addition to characterize leakage and quantify intermediate-value-dependent leakage. In this thesis, SNR analysis is performed alongside TVLA to characterize the leakage observed from the AES cores. Furthermore, we use SNR analysis to compare leakage between the two AES implementations.

Standaert et al. [26] provides an overview of CPA attacks against FPGA implementations of cryptographic algorithms. The authors model the FPGA power consumption as proportional to the number of bit transitions in the device registers. The Hamming distance models the number of bit transitions between consecutive states, motivating the choice to use Hamming distance based models for CPA against FPGA implementations. This is further supported by the ChipWhisperer documentation regarding the CW305 used in this thesis, which notes that register state transitions are an effective target [27]. Furthermore, the documentation states that the ideal leakage model to use on this board is the Hamming distance between the final two states, which is consistent with our findings in Section 6.2. While Hamming weight models can also produce successful CPA attacks on an FPGA in practice, this thesis focuses on Hamming distance based models specifically as they are well motivated for use on FPGA boards by Standaert and the ChipWhisperer documentation and resulted in successful key recovery in our experiments.

Mohajerani et al. [28] provide an extensive evaluation of the SCA resistance of the NIST lightweight finalists. In this article, the Cryptographic Engineering Research Group from George Mason University, also referred to as the GMU team, bench-marked these finalists, also using a CW305 Xilinx Artix-7 XC7A100T FPGA board. The authors concluded that Ascon and Xoodyak performed best for the majority of categories across unprotected and first- to third-order protected implementations. Ascon and Xoodyak demonstrated a notable balance between the bench-marked properties including randomness and speed. Lightweight encryption algorithms like Ascon are more resistant to first-order CPA. In the case of Ascon, the key absorption and state permutations diffuse the key across the entire 320-bit state. After this diffusion, each bit of the output depends on many different key bits. Standard first-order CPA depends on guessing an intermediate value that depends on a portion of the key, which becomes impossible when the intermediate value depends on the entire state. However, various papers show successful side-channel attacks on lightweight ciphers using different techniques such as reinforcement learning and second-order attacks. In a 2020 paper [29], Ramezanpour et al. demonstrate that standard DPA and CPA attacks fail to find the correct key with more than 40K traces. The authors attempted various attacks using the CW305 Artix-7

FPGA board, the same platform used in this thesis. The authors demonstrate how classical CPA and DPA attacks using 40K traces result in incorrect key-guesses with identical or better ranks than the true key-byte. The previously explained side-channel-resistance of Ascon and Xoodyak, and the failed CPA attack displayed in this paper attributed to our decision not to attempt CPA on the lightweight cipher implementations. The authors do, however, show that Side-Channel Analysis with Reinforcement Learning (SCARL), is successful in partial key-recovery using only 24K traces. However, performing such an attack on our platform was outside the scope of this thesis. In 2025, Nguyen et al. [30] performed a successful and practical second-order attack on a masked Ascon implementation. The authors demonstrate how the full 128-bit key could be recovered in 4.7 hours using 360K traces captured from a 32-bit ARMv6 micro-processor running a software Ascon implementation. This paper demonstrates that full key recovery from Ascon is achievable using second-order CPA, though a masked software implementation differs substantially from the unprotected hardware target in this thesis. As the ChipWhisperer leakage models are tailored to AES specifically, a CPA attack on lightweight ciphers would require a carefully constructed selection function. The papers mentioned in this paragraph attributed to our decision to not attempt CPA on Ascon and Xoodyak in this thesis.

Unlike prior work, this thesis assesses the existence of leakage of the PROACT chip prototype, characterizes its leakage, and tests exploitability using standard leakage models, contributing empirical data to the PROACT project.

4 Methodology and Experimental Setup

In this section, we elaborate on the methodology and experimental setup, giving an in-depth overview of the tools and software that are used. First, we explain the setup for capturing the power traces in Section 4.1. Then, we explain the various sets of power traces captured under various configurations in Section 4.2. Lastly, we briefly elaborate on the scripts used to analyze the captured traces in Section 4.3. All scripts, firmware, and traces discussed in this section are available on the LIACS GitLab server [6]. In this thesis, we present and discuss pseudo-code for the critical parts of the scripts.

4.1 Trace Capturing Setup

We give an overview of the devices used in the setup, a diagram of this setup is provided in Figure 11. The setup consists of a host (PC), the ChipWhisperer Husky, and the CW305 with the PROACT chip prototype loaded in the FPGA.

[6] https://git.liacs.nl/s3813606/thesis-2026-project-files/

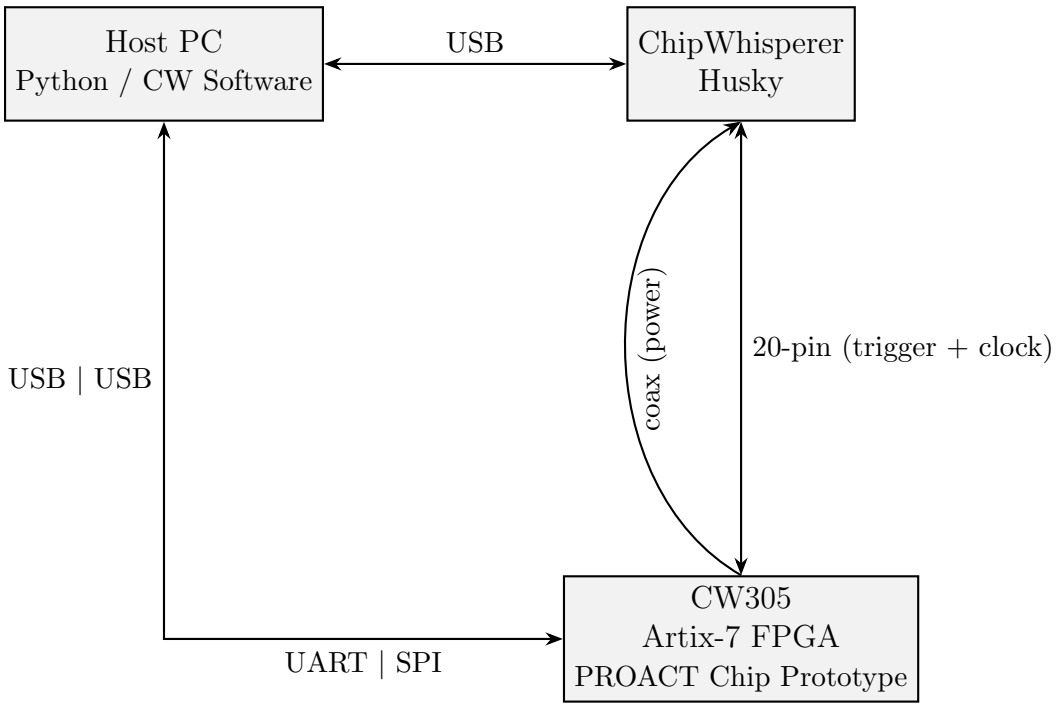

Figure 11: Trace capturing setup

The host is connected to the CW305 using 2 USB connections, one for the UART connection, and one for the SPI bridge. Furthermore, the host is connected to the Husky over USB. The Husky is connected to the CW305 using a 20-pin connector and a coax cable for power-trace capturing.

4.1.1 Target

As mentioned in Section 2.3, we use the CW305 target platform with the PROACT chip prototype loaded onto the Artix-7 XC7A100T FPGA as the target. The clock frequency on the CW305 is set to 50 MHz and we use the onboard PLL for clock generation. The target is programmed with the PROACT chip bit-stream over USB using ChipWhisperer's Python API. The board is powered over USB from the host. Communication between the host and the controller core is performed over UART using our custom Python scripts and the `pyserial` library.

The controller of the target is programmed to run our custom C firmware that is used to receive data from the host and trigger the co-processors. This C firmware runs a loop in which it awaits *messages* from the host over UART. These messages start with a command byte, telling the controller what to do and how to interpret the payload if applicable. The payload contains additional information such as the cryptographic key or plaintext. When the controller is commanded to start encryption, the hardware fires a capture signal that is picked up by the capturing device through the 20-pin connector. This firmware allows for flexibility, i.e., we can command the board to execute multiple encryption algorithms using different inputs, all using a single python code-block execution. Algorithm 1 presents pseudocode for the command processing loop of the firmware.

Algorithm 1: Target Command Processing Loop

```
while true do
    Receive command c;
    switch c do
        case Mode selection do
            Set encryption mode;
        case Key command do
            Receive key;
        case Plaintext command do
            Receive plaintext;
        case Nonce command do
            Receive nonce;
        case Associated data command do
            Receive associated data;
        case Ready command do
            Wait for signal and execute selected encryption algorithm;
```

4.1.2 Capturing Device

As a capturing device, we use the CW Husky. The ADC sampling rate is set to 200 MHz, this is four times higher than the clock frequency of the target, giving us 4 samples per clock cycle. The Husky is connected to the target using a 20-pin connector. This connector is used for clock synchronization and for firing the trigger. For the clock source, we use the `extclk` mode, making the Husky lock to the target clock. Furthermore, a coax cable is connected between the Husky and the CW305 board for the capturing of power consumption. The Husky is also connected to the host over USB for communication and programming. We set the sample count for the Husky to 5000 samples per trace, which is enough to capture the entire encryption window. During analysis and CPA, we trim these windows to isolate the exact encryption windows.

4.1.3 Host

As a host, we use a PC running Python 3.14.4 with the ChipWhisperer 6.0.0 library. The host runs a custom python script in which we call our capturing loop using different sets of parameters. The loop configures the controller with the passed settings and encryption parameters. The flexibility of this loop allows us to capture all traces for all our sets in a single code-block execution. Algorithm 2 presents the pseudocode for the trace capturing loop.

Algorithm 2: Interleaved Fixed-vs-Random Trace Capture Loop

Input: Varying parameter $p \in \{\texttt{plaintext}, \texttt{nonce}, \texttt{AD}\}$
Input: Target algorithm $a \in \{\texttt{AES}, \texttt{AES2}, \texttt{Ascon}, \texttt{Xoodyak}\}$
Input: Number of traces N
while *fixed traces* $< N$ ***or*** *varying traces* $< N$ **do**
    Select fixed or varying group
    if *varying group selected* **then**
        Generate random value for parameter p
    Configure target with mode and values
    Capture power trace
    if *capture successful* **then**
        Store trace in corresponding set
for $a \in \{\textit{AES}, \textit{AES2}\}$ **do**
    Acquire traces using $p = \texttt{plaintext}$ and algorithm a
for $p \in \{\textit{plaintext}, \textit{nonce}, \textit{AD}\}$ **do**
    for $a \in \{\textit{Ascon}, \textit{Xoodyak}\}$ **do**
        Acquire traces using parameter p and algorithm a

4.2 Power-trace Sets

We capture sets of traces for four algorithmic co-processors under various experimental setups. Our setups will each use a target algorithm $a \in \{\texttt{AES}, \texttt{AES2}, \texttt{Ascon}, \texttt{Xoodyak}\}$ and a varying parameter $p \in \{\texttt{plaintext}, \texttt{nonce}, \texttt{AD}\}$[7] for a total 8 experimental setups, each resulting in a set of fixed parameter traces and a set of varying parameter traces. For TVLA, we require a fixed group A, and a randomly varying parameter group B. Therefore, we capture a total of $8 \times 2 \times 10000 = 160000$ traces. We caputure 10K traces for each algorithm's group A using a fixed key and fixed plaintext, and if applicable, fixed nonce and associated data. Group B has one of the non-key variables varying each iteration. This allows us to detect the following:

1. Plaintext-dependent power-consumption on AES1, AES2, Ascon, and Xoodyak;
2. Nonce-dependent power-consumption on Ascon and Xoodyak;
3. AD-dependent power-consumption on Ascon and Xoodyak.

Note that the total amount of traces could be reduced by discarding the overlapping fixed group power-trace sets. However, as mentioned in Section 3, this could cause systemic bias from variables such as temperature changes in the measurement equipment [23]. To reduce systemic bias, Goodwill et al. recommend capturing the traces in an interleaved manner. Thus, we take the following two measures:

1. We recapture the traces of the fixed group for each experiment. We run a total of eight experimental setups, each capturing two groups of 10K power traces.

[7] Nonce and AD are not applicable for the AES algorithms and thus AES will not be paired with these parameters.

2. We use random interleaving for the capturing of the trace groups, i.e., we randomly assign each trace capture iteration to either group A or B with equal probability. When a group reaches 10K traces, all remaining iterations are assigned to the remaining group.

In this thesis, we refer to the 8 experimental setups as E1-E8. An overview of the experimental setups is given in Table 1.

ID	Algorithm	Varying parameter
E1	AES1	Plaintext
E2	AES2	Plaintext
E3	Ascon	Plaintext
E4	Ascon	Nonce
E5	Ascon	Associated data
E6	Xoodyak	Plaintext
E7	Xoodyak	Nonce
E8	Xoodyak	Associated data

Table 1: Overview of the eight experimental setups.

The power traces are captured using our custom Python capturing script. The capture loop is executed 8 times, once for each experimental setup. For each algorithm, a 16-byte key and plaintext is used. When applicable, a 16-byte nonce and associated data is used.

For performing CPA on the AES co-processors, the varying plaintext traces from E1 and E2 are reused. The ChipWhisperer Capture API allows us to store the key, ciphertexts, and plaintexts along with the traces. To reduce overhead and code complexity, ciphertexts are computed on the host using a verified reference AES software implementation provided by the `PyCryptodome` [31] Python library rather than captured directly from the target. To verify the consistency between the [31] output and the target output, a set of 10 randomly selected ciphertexts are verified for both AES co-processors, after which we confirm identical results. Furthermore, successful key recovery in the subsequent CPA section confirms the correctness of the stored data.

4.3 Analysis of the Power Traces

We deploy three analysis strategies on the traces. For each strategy, the full, but isolated window of co-processor operation is used. To determine the window in which the co-processor is operational, we count the clock cycles that occur between the firing of the trigger signal and the co-processor's done state bit toggle. Using our sampling rate, we are able to convert these cycle counts to a sample window. The cycle counts and corresponding sample window sizes are shown in Table 2.

Co-processor	Cycles	Samples
AES1	36	144
AES2	36	144
Xoodyak	55	220
Ascon	77	308

Table 2: Encryption cycle and sample counts for each co-processor.

4.3.1 Test Vector Leakage Assessment

To perform TVLA, we use a custom script that automates the data-to-plot pipeline. We use the `stats.ttest_ind` method from the `SciPy` Python library [32] with `equal_var=False` to run Welch's t-test. The function takes two `NumPy` [33] arrays containing the power traces of groups A and B. We plot the results using the `Matplotlib` library [34].

To automate TVLA analysis on our trace sets, the script contains a set of trace set configurations. The user can add, remove, or change configurations. Running the script will automatically perform TVLA analysis on all the registered configurations. The script outputs, for each configuration, a summary of the statistics, and a plot with the visualized results. The script allows for simple modification and re-usability in future projects.

4.3.2 Signal-to-Noise Ratio

For SNR Analysis, we run a custom Python script to calculate the SNR using `NumPy` arrays. The result is another `NumPy` array, allowing for a simple pipeline from power-trace set to plot. The results are plotted using `Matplotlib`. The intermediate values are directly computed using the selection functions provided by the leakage models discussed in Section 2.4.

Similar to the TVLA script, the SNR script automates the process for a set of configurations. Running the script will output plots with the visualized results for each registered configuration.

4.3.3 Correlation Power Analysis

For Correlation Power Analysis, the ChipWhisperer Python library is used. We perform CPA using a subset of ChipWhisperer's collection of standard leakage models on windowed power traces. For our custom leakage model, we use the ChipWhisperer API to register it as a leakage model, allowing us to use it and interpret the results identically.

As with the SNR and TVLA scripts, the CPA script uses a configuration-set-based approach. The user can add configurations of trace sets and target leakage models. Running the script will automatically perform CPA on each registered configuration and output the statistics for each attack.

5 Leakage Detection Using TVLA (RQ1)

We run our TVLA analysis for each experimental setup (E1 to E8) and report the statistics. Each reported plot figure in the following subsections contains two types of subplots:

1. Plots displaying example power traces from the two trace sets. The y-axis displays the power amplitude and the x-axis the sample index. The purpose of these plots are to give the reader a visual representation of the power consumption patterns.
2. A plot displaying the calculated t-values. The y-axis displays the t-statistic and the x-axis the sample index. Furthermore, this plot displays the clock cycle number at the top, consistent with the four samples per cycle as discussed in Section 4. Furthermore, the plots contain dashed lines to display the 4.5 t-value threshold.

As discussed in Section 2, TVLA is prone to false positives. Therefore, isolated, marginal threshold crossings should be interpreted with caution.

We give a summary of all the results in Table 3. Then, we discuss each experiment individually in the following subsections. In the table, we show the following metrics:

1. the experimental setup number, together with the corresponding co-processor and varying parameter,
2. the presence of leakage,
3. the maximum observed $|t|$ value,
4. the index of the maximum,
5. the leakage window (first and last sample indices),
6. the total amount of samples with a t value exceeding the threshold within the encryption window,
7. the percentage of leaking samples.

Experiment	Leakage	Max $\|t\|$	Max $\|t\|$ index	Window [first, last]	Total	Leakage %
E1: AES1 - PT	Yes	87.35	43	[13, 143]	104	72.22
E2: AES2 - PT	Yes	193.08	65	[2, 143]	100	69.44
E3: Asc. - PT	Yes	59.50	198	[121, 307]	131	42.53
E4: Asc. - nonce	Yes	92.04	94	[5, 307]	259	84.09
E5: Asc. - AD	Yes	106.68	118	[0, 307]	247	80.19
E6: Xoo. - PT	Yes	68.01	154	[141, 219]	72	32.73
E7: Xoo. - nonce	Yes	103.61	59	[5, 219]	193	87.73
E8: Xoo. - AD	Yes	185.64	74	[0, 219]	196	89.09

Table 3: TVLA results summary - all experiments.

From the results shown in Table 3, we conclude that TVLA shows leakage for each of the experimental setups. E2 resulted in the strongest leakage, with a maximum $|t|$ value of 193.08. However, we

conclude that AES1 displays a higher leakage percentage. For both the Ascon and Xoodyak co-processors, the varying AD experiments showed the highest maximum $|t|$ value and leakage percentage among the varying parameters, and in both cases, the plaintext showed the lowest maximum $|t|$ value and the lowest leakage percentage.

5.1 E1: AES1 - vary plaintext

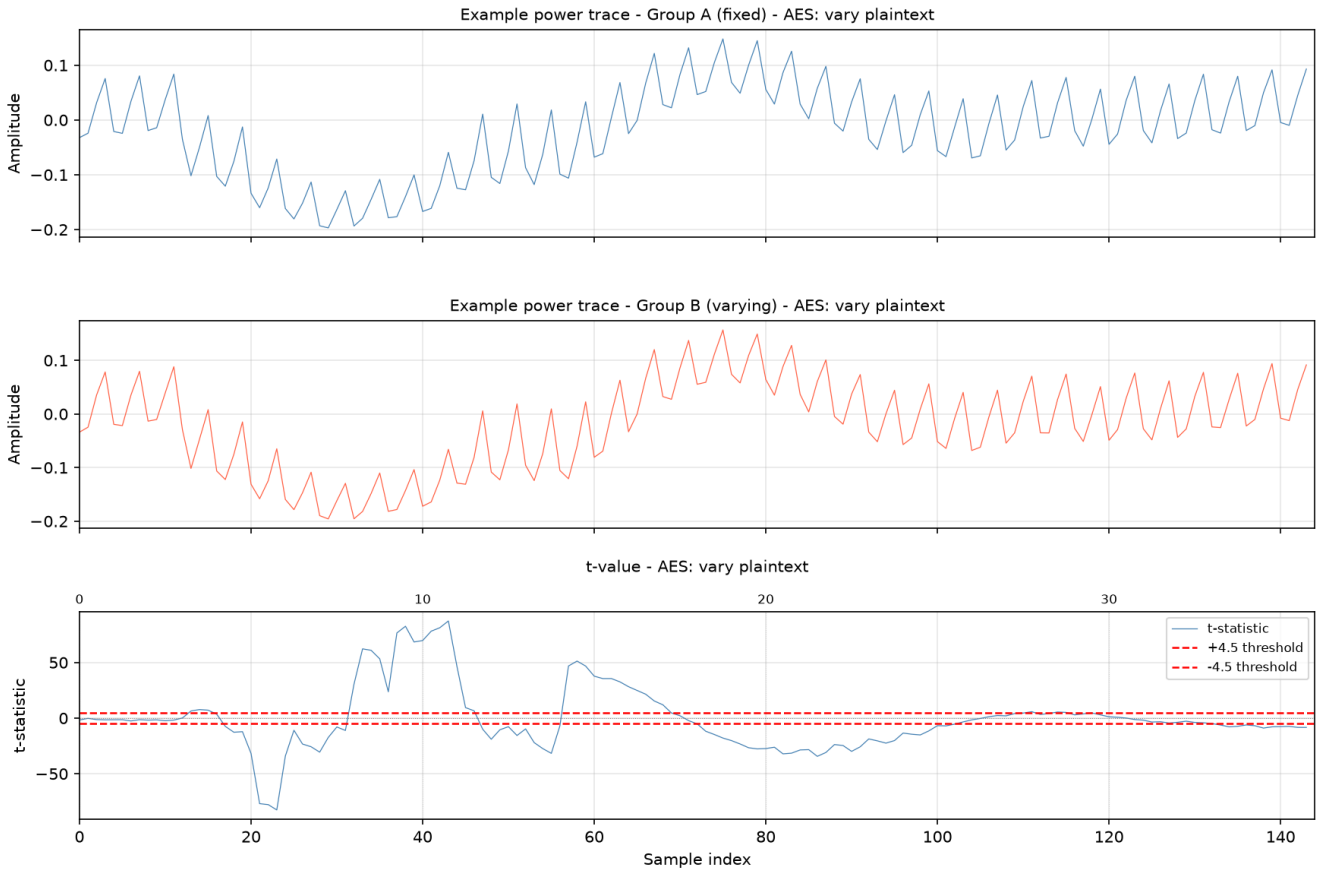

Figure 12: AES1 - fixed vs varying plaintext - example power traces (top) and t-values (bottom).

TVLA was performed on traces that were captured during cryptographic operations using a 16-byte key and a 16-byte plaintext. In this experiment, trace set B was captured using random 16-byte plaintext input values.

The TVLA results for E1 are displayed in Figure 12. TVLA on E1 confirms statistically significant plaintext-dependent leakage. The results show 104 sample points with threshold exceeding $|t|$ values in the range [13, 143]. The maximum reported $|t|$ value is 87.35 at sample index 43, which is significantly higher than the 4.5 threshold.

We can observe from the plot that the most significant leakage elevations lie roughly between the sample range 20-100. The first t spike occurs roughly at sample index 20 (clock cycle 5). Our SNR analysis in Section 6.1 confirms that this spike overlaps with the first AES rounds. The spike at sample index 60 (clock cycle 15) overlaps with the last round's SNR spike.

Using the SNR results presented in Section 6.1 and A, we confirm that the 10 AES rounds occur roughly in the range [20-60], consistent with 1 clock cycle per AES round given our sampling rate. The alignment between TVLA and SNR peaks supports a causal relationship with AES round

switching activity rather than measurement artifacts. From these results, we conclude that the leakage between clock cycles 5-15 can likely be attributed to the 10 AES rounds.

We observe leakage outside the 10 AES rounds, however, the $|t|$ values are relatively low and can likely be attributed to computational overhead within the co-processor.

5.2 E2: AES2 - vary plaintext

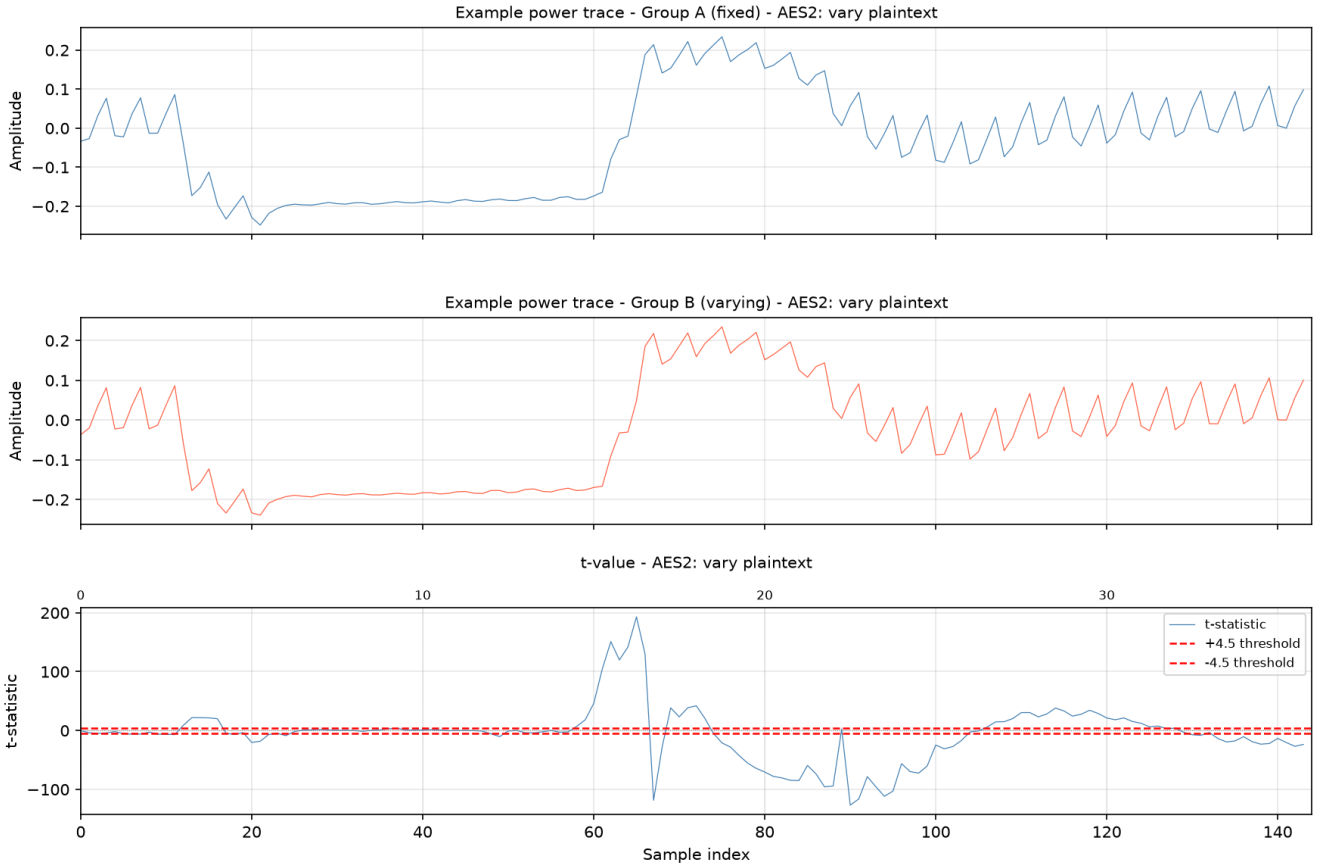

Figure 13: AES2 - fixed vs varying plaintext - example power traces (top) and t-values (bottom).

TVLA was performed on traces that were captured during cryptographic operations using a 16-byte key and a 16-byte plaintext. In this experiment, trace set B was captured using random 16-byte plaintext input values.

The TVLA results for E2 are displayed in Figure 13. TVLA on E2 confirms statistically significant plaintext-dependent leakage. The results show 100 sample points with threshold exceeding $|t|$ values in the range [2, 143]. The maximum reported $|t|$ value is 193.08 at sample index 65. Notably, the amount of leakage is slightly lower compared to AES1 (100 vs 104). However, the maximum $|t|$ for AES2 is significantly higher, suggesting that AES2 exhibits substantially stronger leakage.

Similar to the results from E1, we observe overlap between the TVLA and SNR results. Using the SNR results presented in Section 6.1 and Appendix A, we confirm that the first AES round occurs

roughly at sample index 20 (clock cycle 5). The last round occurs roughly at sample index 60 (clock cycle 15). This alignment suggests, similarly to E1, that the 10 AES rounds occur roughly between clock cycles 5-15.

For AES2, we can observe from Figure 31, we can observe that last round SNR values are elevated between samples 80-100 as well. However, as the last round's intermediates are a bijection mapping to the ciphertext, resulting in identical SNR grouping, the spike can likely be attributed to some register write of the ciphertext.

Remarkably, AES1 (E1) shows significantly more leakage in the 20-60 range, suggesting that the LUT-based S-box implementation of AES1 results in substantially more data-dependent leakage than AES2's bit-wise implementation. Furthermore, we can observe from Figure 13 that the power amplitude variance is relatively low between sample points 20-60, suggesting that the LUT-based S-box implementation causes the power amplitude to remain relatively fixed during the AES rounds.

5.3 E3: Ascon - vary plaintext

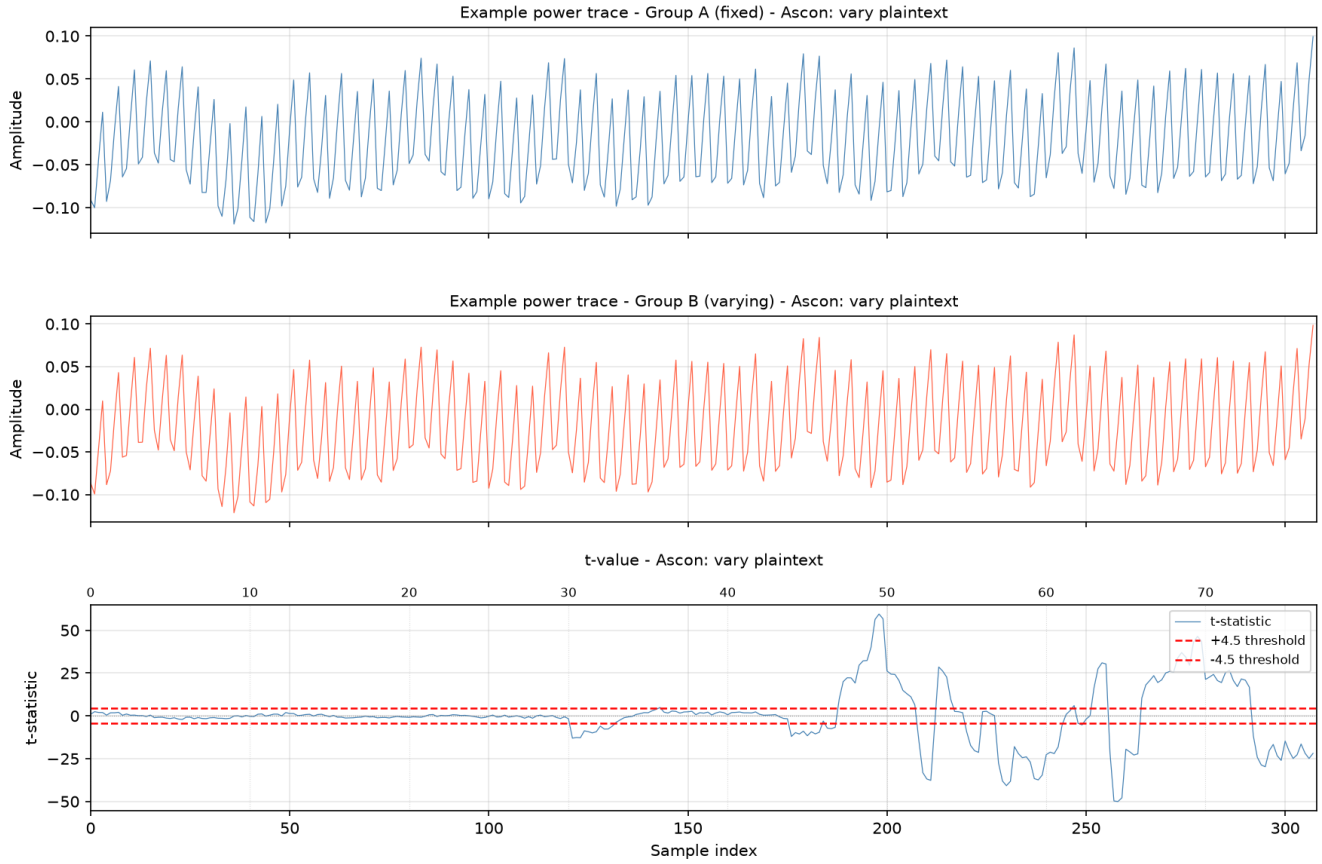

Figure 14: Ascon - fixed vs varying plaintext - example power traces (top) and t-values (bottom).

TVLA was performed on traces that were captured during cryptographic operations using a 16-byte key, a 16-byte nonce, a 16-byte associated data and a 16-byte plaintext. In this experiment, trace set B was captured using random 16-byte plaintext input values.

The TVLA results for E3 are displayed in Figure 14. TVLA on E3 confirms statistically significant plaintext-dependent leakage. The results show 131 sample points with threshold exceeding $|t|$ values in the range [121, 307]. The maximum reported $|t|$ value is 59.50 at sample index 198. We can observe that the most drastic t spikes occur roughly within the last 25 clock cycles.

As we did not perform SNR analysis on Ascon and Xoodyak, reasoning about specific sample points becomes more difficult. However, by relating the observed leakage patterns to the expected execution flow of the encryption algorithms, we can make informed inferences about the underlying operations. Furthermore, precise mapping between clock cycles and specific round operations would require hardware-level analysis which falls outside the scope of this thesis but remains an opportunity for future work.

Comparing the results displayed in Figure 14 to the Ascon cryptography flow displayed in Figure 5, we can observe that the plaintext encryption and finalization phases likely occur roughly between sample points 200-300 (clock cycles 50-75), as that is when the co-processor exhibits plaintext-dependent power consumption. Given the known structure of the encryption algorithm, this interval likely coincides with the state update operations involving plaintext-dependent transformations.

The consistency between the Ascon encryption flow and the observed leakage window supports a causal relationship rather than indicating measurement artifacts in this experiment.

5.4 E4: Ascon - vary nonce

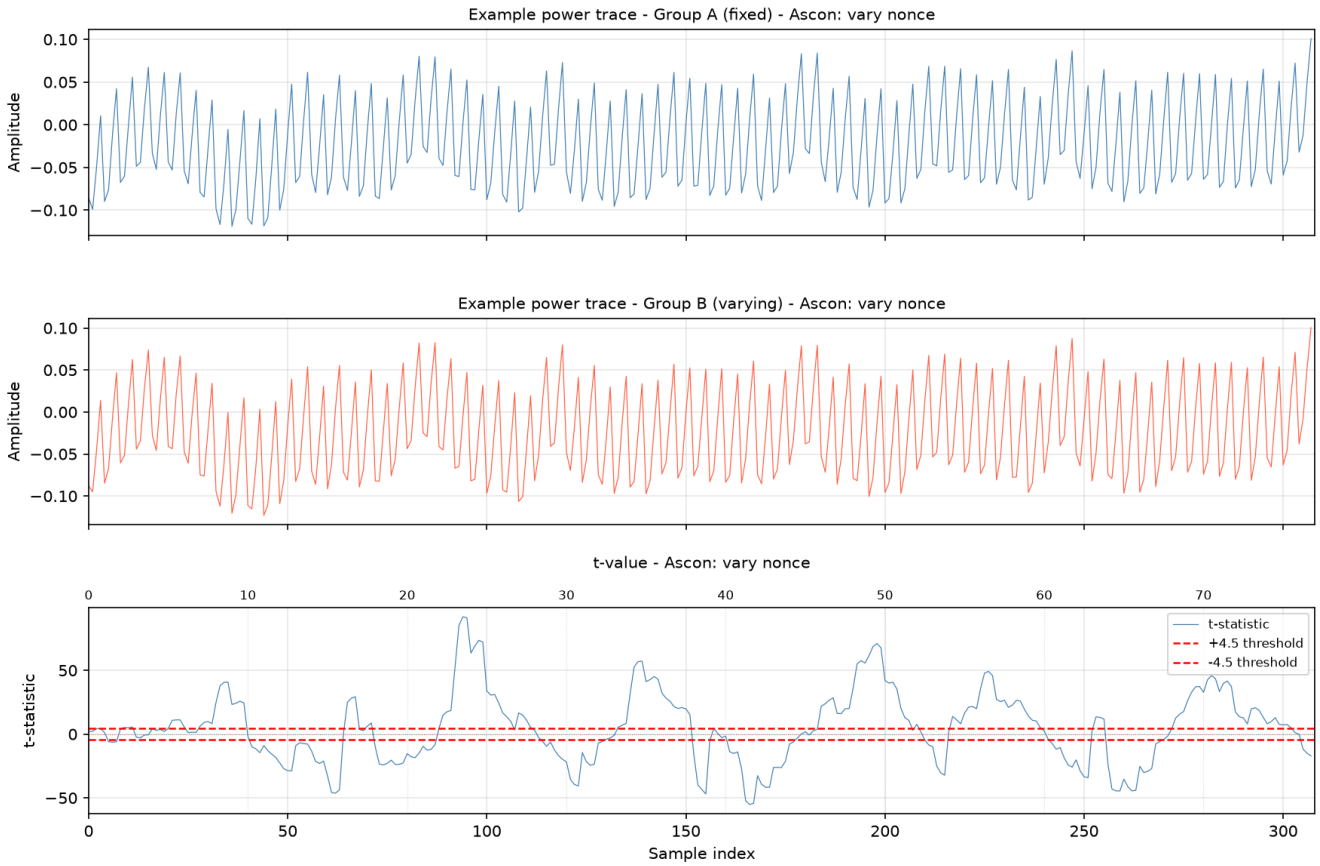

Figure 15: Ascon - fixed vs varying nonce - example power traces (top) and t-values (bottom).

TVLA was performed on traces that were captured during cryptographic operations using a 16-byte key, a 16-byte nonce, a 16-byte associated data and a 16-byte plaintext. In this experiment, trace set B was captured using random 16-byte nonce input values.

The TVLA results for E4 are displayed in Figure 15. TVLA on E4 confirms statistically significant nonce-dependent leakage. The results show 259 leaking samples in the range [5, 307]. The maximum reported $|t|$ value is 92.04 at sample index 94.

Considering the Ascon encryption flow, we expect nonce-dependent leakage to occur near the start of the co-processor's operation window. During the initialization phase, the nonce is absorbed into the state. Consequently, state update operations involving the nonce are expected to occur during this first encryption phase.

As with E3, the observed leakage window aligns with the expected encryption flow, supporting the hypothesis that the leakage is algorithmic-dependent rather than an artifact from the measurement setup.

5.5 E5: Ascon - vary associated data

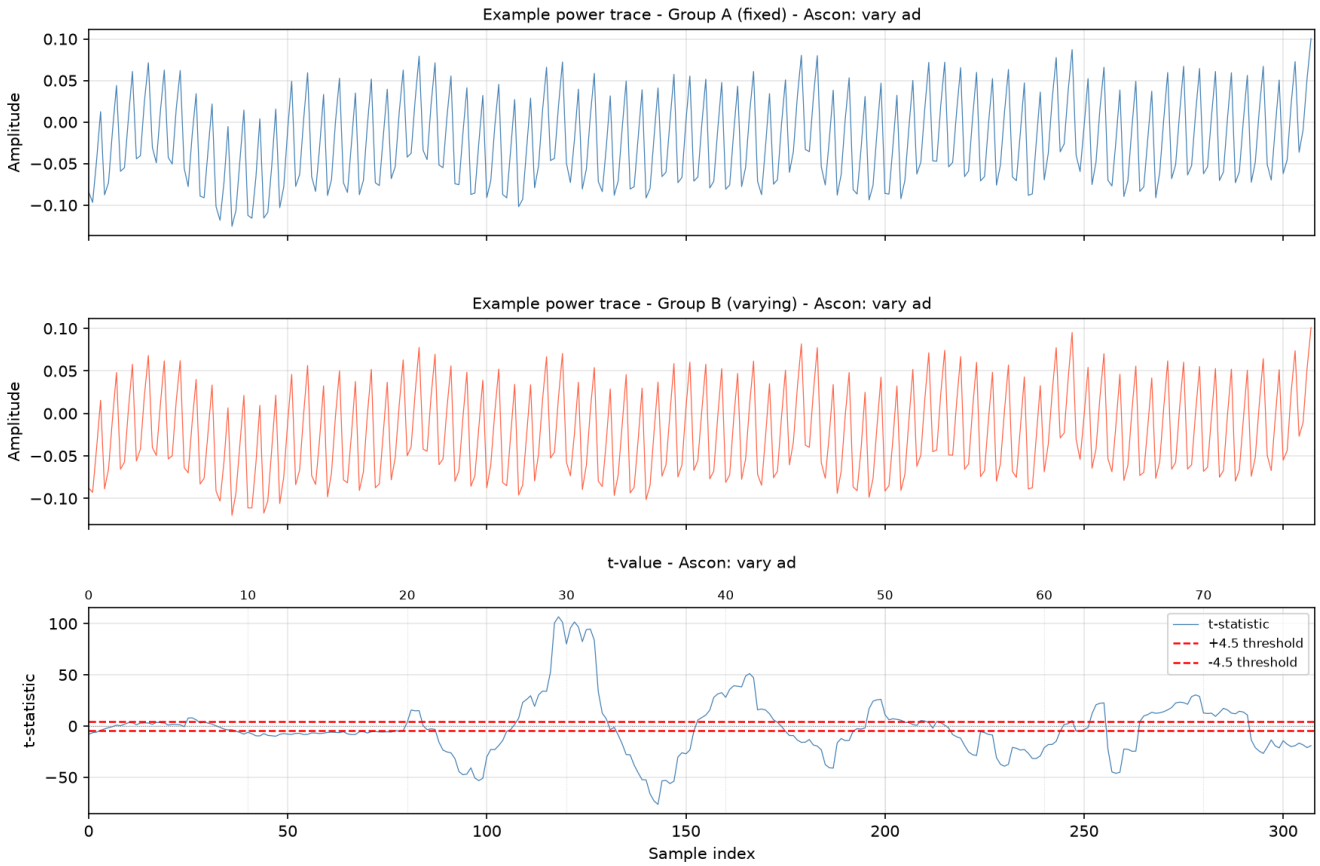

Figure 16: Ascon - fixed vs varying associated data - example power traces (top) and t-values (bottom).

TVLA was performed on traces that were captured during cryptographic operations using a 16-byte key, a 16-byte nonce, a 16-byte associated data and a 16-byte plaintext. In this experiment, trace set B was captured using random 16-byte associated data input values.

The TVLA results for E5 are displayed in Figure 16. TVLA on E5 confirms statistically significant AD-dependent leakage. The results show 247 leaking samples in the range [0, 307]. The maximum reported $|t|$ value is 106.68 at sample index 118.

As with E3 and E4, we compare the leakage displayed in Figure 16 with the Ascon encryption flow. We expect the nonce to be the first variable to display leakage, followed by the associated data, and finally the plaintext. The results displayed in Figure 16 confirms this hypothesis. Altough t-values slightly exceed the threshold immediately, we can observe substantial leakage starting roughly at sample point 80.

From the results of E3, E4, E5, and the encryption flow displayed in Figure 5, we can conclude that the leakage profile for the Ascon co-processor is consistent with the expected Ascon cryptography flow. The results display a clear phase separation, showing distinct leakage windows for the plaintext-

, nonce-, and associated-data-dependent leakage. The clear phase separation and consistent leakage order supports the hypothesis that the leakage is algorithmic-dependent.

5.6 E6: Xoodyak - vary plaintext

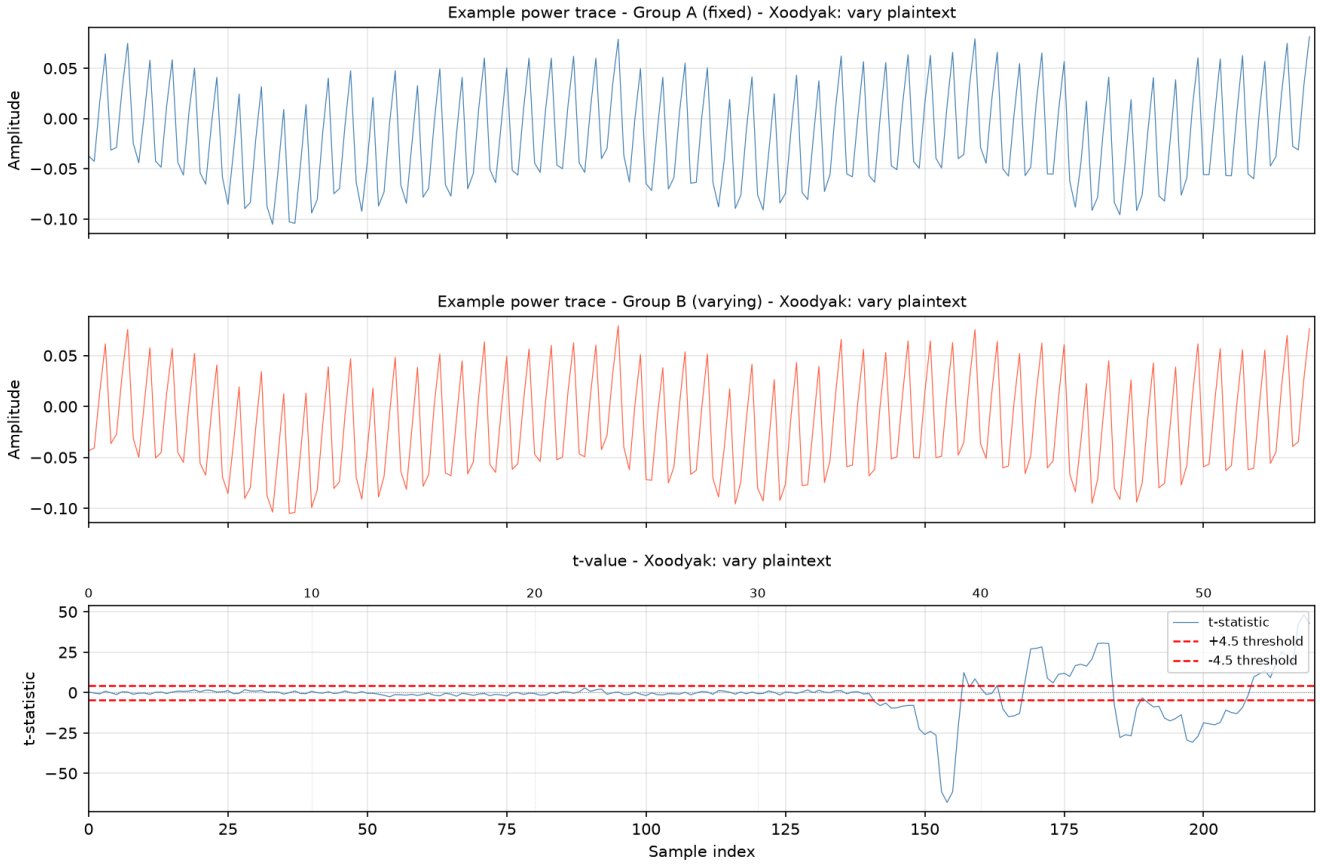

Figure 17: Xoodyak - fixed vs varying plaintext - example power traces (top) and t-values (bottom).

TVLA was performed on traces that were captured during cryptographic operations using a 16-byte key, a 16-byte nonce, a 16-byte associated data and a 16-byte plaintext. In this experiment, trace set B was captured using random 16-byte plaintext input values.

The TVLA results for E6 are displayed in Figure 17. TVLA on E6 confirms statistically significant plaintext-dependent leakage. The results show 72 leaking samples in the range [141, 219]. The maximum reported $|t|$ value is 68.01 at sample index 154.

Similar to the Ascon results, we may expect the leakage window to be consistent with the Xoodyak cryptographic flow. At a high level, Xoodyak follows a similar cryptography flow, where the nonce is absorbed first, followed by the associated data, and finally the plaintext is processed to produce the output. For the specific details of the Xoodyak cryptography flow, we refer the reader to the Xoodyak specification [11].

In Figure 17, we observe that the leakage window lies roughly between the 140-220 range, suggesting a window of roughly 20 clock cycles with transformations depending on the plaintext.

5.7 E7: Xoodyak - vary nonce

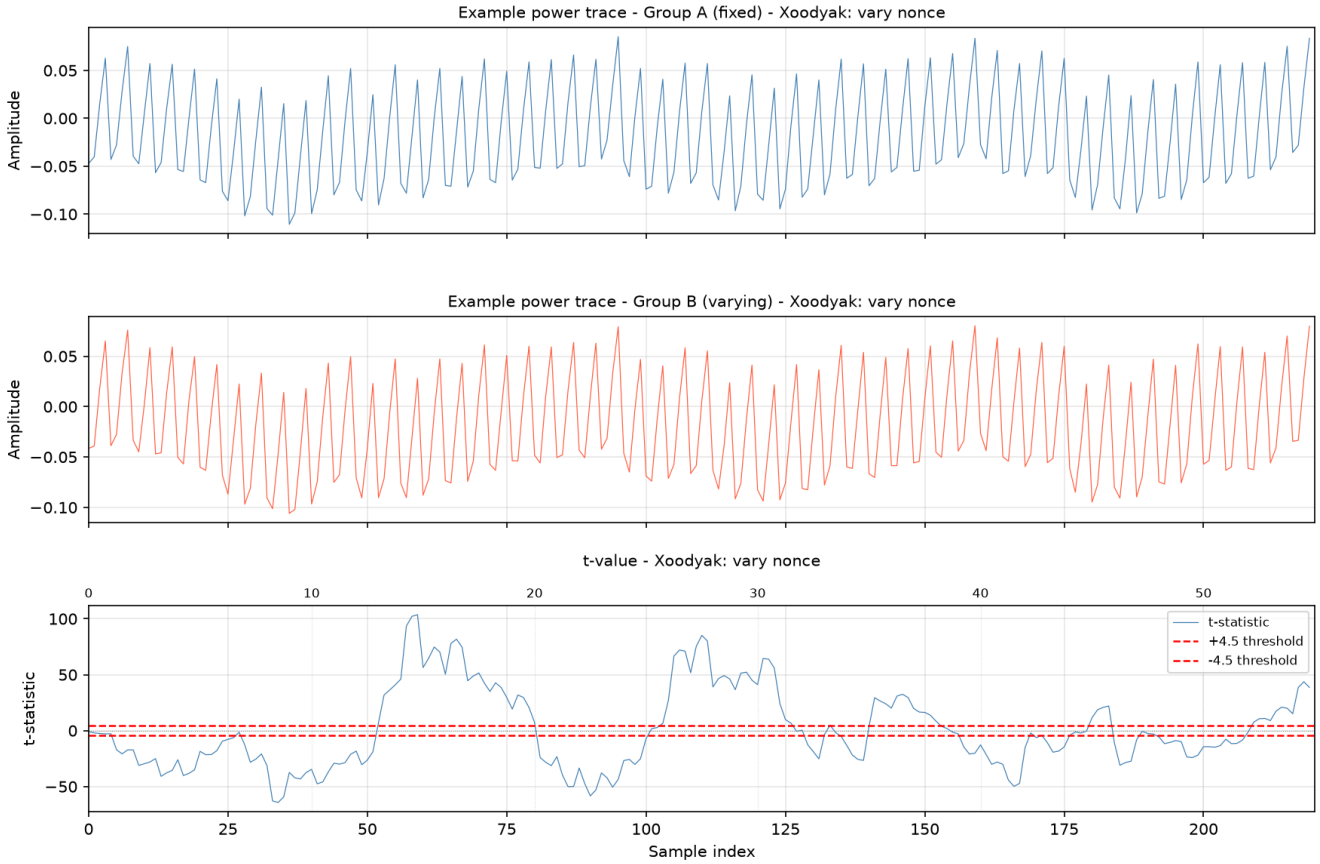

Figure 18: Xoodyak - fixed vs varying nonce - example power traces (top) and t-values (bottom).

TVLA was performed on traces that were captured during cryptographic operations using a 16-byte key, a 16-byte nonce, a 16-byte associated data and a 16-byte plaintext. In this experiment, trace set B was captured using random 16-byte nonce input values.

The TVLA results for E7 are displayed in Figure 18. TVLA on E7 confirms statistically significant nonce-dependent leakage. The results show 193 leaking samples in the range [5, 219]. The maximum reported $|t|$ value is 103.61 at sample index 59.

Similar to E4, we observe that nonce-dependent leakage occurs almost immediately. The nonce is absorbed into the state before the associated data and plaintext are processed, hence the near immediate leakage is expected. The distinguishability between E6 and E7 being consistent with the Xoodyak cryptographic flow supports the hypothesis that the leakage is algorithmic-dependent.

5.8 E8: Xoodyak - vary associated data

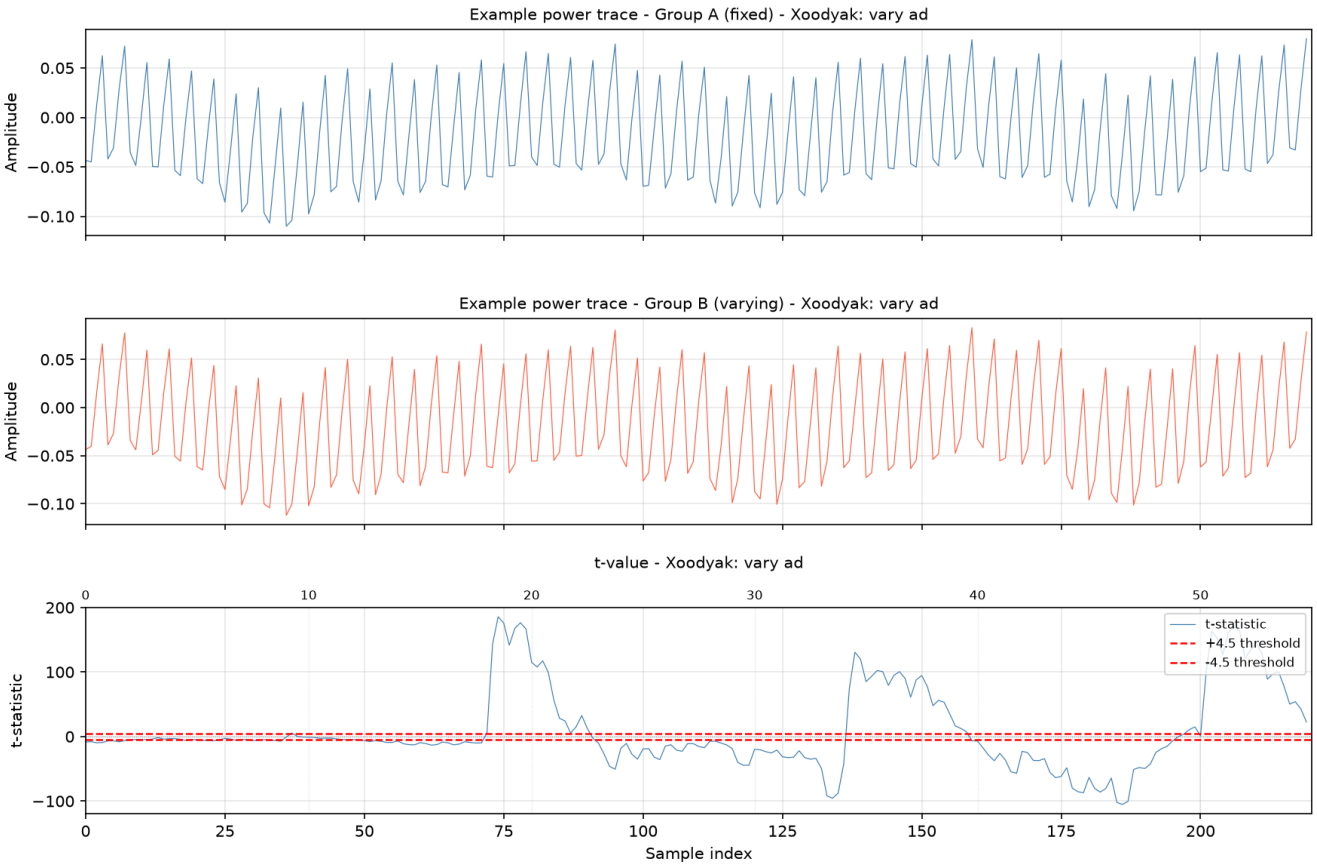

Figure 19: Xoodyak - fixed vs varying associated data - example power traces (top) and t-values (bottom).

TVLA was performed on traces that were captured during cryptographic operations using a 16-byte key, a 16-byte nonce, a 16-byte associated data and a 16-byte plaintext. In this experiment, trace set B was captured using random 16-byte associated data input values.

The TVLA results for E8 are displayed in Figure 19. TVLA on E8 confirms statistically significant AD-dependent leakage. The results show 196 leaking samples in the range [0, 219]. The maximum reported $|t|$ value is 185.64 at sample index 74.

As with Ascon, we compare the results of E6, E7, and E8 and observe distinct leakage windows, suggesting a real algorithmic-dependent pattern. We can observe that nonce-dependent leakage occurs throughout the entire window of operation, while the associated data is absorbed at a later stage and thus starts causing data-dependent leakage at a later point. Finally, the plaintext is processed to produce the ciphertext, causing the plaintext-dependent leakage window to start last.

Similar to Ascon, we conclude that the leakage windows of E6, E7, and E8 are consistent with the Xoodyak algorithm, supporting the hypothesis that the leakage is algorithmic-dependent.

5.9 Conclusions (RQ1)

For RQ1, we conclude that all four co-processors of the PROACT chip prototype leak statistically significant data-dependent power consumption. Formally, for any input parameter $p \in P$ that is not the key-parameter p_k, TVLA on the power trace set T_E shows statistically significant p-dependent power leakage on the PROACT chip prototype implementation of E. This holds for all $E \in \{\text{AES1, AES2, Ascon, Xoodyak}\}$. For each experimental setup, we reject H_0 and accept H_a.

Remarkably, we observed that AES1 shows substantially stronger leakage between samples 20-60 when compared to AES2, suggesting that the LUT-based S-box implementation shows stronger leakage on our setup. However, AES2 showed significantly stronger leakage around sample 60, which overlaps with the last round's computations.

Noteworthy is that for both Ascon and Xoodyak, the plots corresponding to their testing parameters show clear phase separation consistent with their cryptographic flow. For both Ascon and Xoodyak, the associated data showed both the most significant leakage and the highest number of samples above the 4.5 threshold.

In Section 2, we discussed several drawbacks of TVLA analysis, one of which being the possibility of producing false positives. Therefore, isolated and marginal threshold crossings observed in Section 5 should be interpreted with caution. However, the maximum $|t|$ values reported in Section 5 exceed the threshold by a significant amount. Furthermore, for the AES implementations, the TVLA results align with the SNR analysis discussed in Section 6.1, and for the lightweight ciphers, the leakage windows show distinct leakage windows that align with the expected algorithmic procedure. Therefore, the results indicate that the co-processors are likely to exhibit algorithm-dependent leakage.

6 AES - Exploitability Analysis (RQ2)

In this section, we show that CPA can be used to recover the full encryption key for PROACT's AES co-processors. We first quantify each model's leakage strength using SNR analysis, computing the max and mean SNR across all key-bytes using the selection functions of each leakage model discussed in Section 2.4. We compute the mean and max across all 16 bytes for each leakage model. Leakage models whose intermediate values cause data-dependent variance show an increase of both mean and max SNR, while models that fail to find correlation show no distinguishable increase above the noise floor.

As there is no universal threshold for SNR, we will not use the results to guide this our initial CPA attack. Instead, we use the results to characterize the leakage of the co-processors and compare how leakage differs between the two implementations. However, we do perform an additional SNR-guided CPA attack in Section 6.3 to confirm whether the SNR results align with the exploited leakage windows.

We present the CPA results on the full encryption window in Section 6.2.

6.1 SNR Analysis

In Section 2.4, we discussed the six leakage models that are relevant to our setup. For each leakage model, we compare the SNR results between AES1 and AES2, using the maximum mean value as the comparison metric. The results are presented in Table 4 and a short discussion of the results is provided. Furthermore, a limited selection of plots are presented and discussed in this section (refer to Appendix A for the full set of plotted SNR results). Note that due to SNR's grouping property, some intermediates result in identical groups and thus identical results.

Leakage Model	AES1	AES2
`last_round_state_diff`	0.0306	0.0322
`last_round_state_diff_alternate`	0.0364	0.0370
`sbox_output`	0.0525	0.0634
`sbox_in_out_diff`	0.0395	0.0477
`round_1_2_state_diff_text`	0.0556	0.0293
`round_1_2_state_diff_keymix`	0.0556	0.0293
`round_1_2_state_diff_sbox`	0.0292	0.0291
`custom_1_2_state_diff`	0.0492	0.0295

Table 4: Peak mean SNR per leakage model for AES1 and AES2.

We can observe from Table 4 that AES1 has a significantly higher mean SNR peak for `round_1_2_state_diff_text`, `round_1_2_state_diff_keymix`, and `custom_1_2_state_diff`; AES2 shows no meaningful elevation for these models. Neither model shows SNR elevation above the noise floor for `round_1_2_state_diff_sbox`. AES2 shows a stronger spike for both `sbox_output` and `sbox_in_out_diff`, which may reflect AES2's bit-wise S-box implementation.

To characterize the leakage, we can reason about the used intermediate values and the observed SNR spikes. A high SNR spike using some intermediate value computed in some round or operation, suggests that that round or operation is performed within the sample point range of the observed spike.

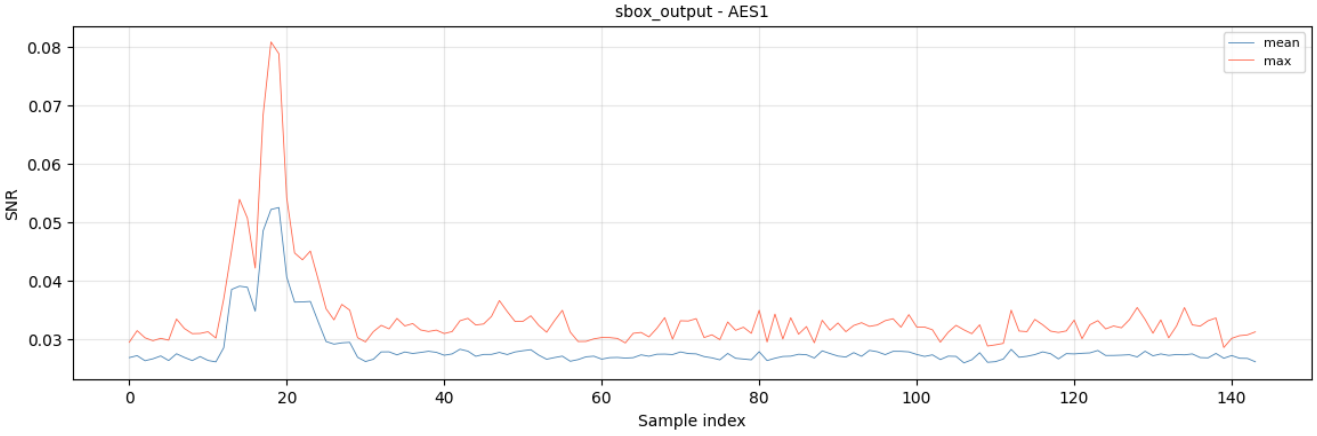

Figure 20: SNR - `sbox_output` - AES1

In Figure 20, we can observe a spike in SNR roughly around sample 20. The SNR spike and the intermediate values corresponding to the first round's S-box output suggests that the first AES round is computed roughly at the fifth clock cycle.

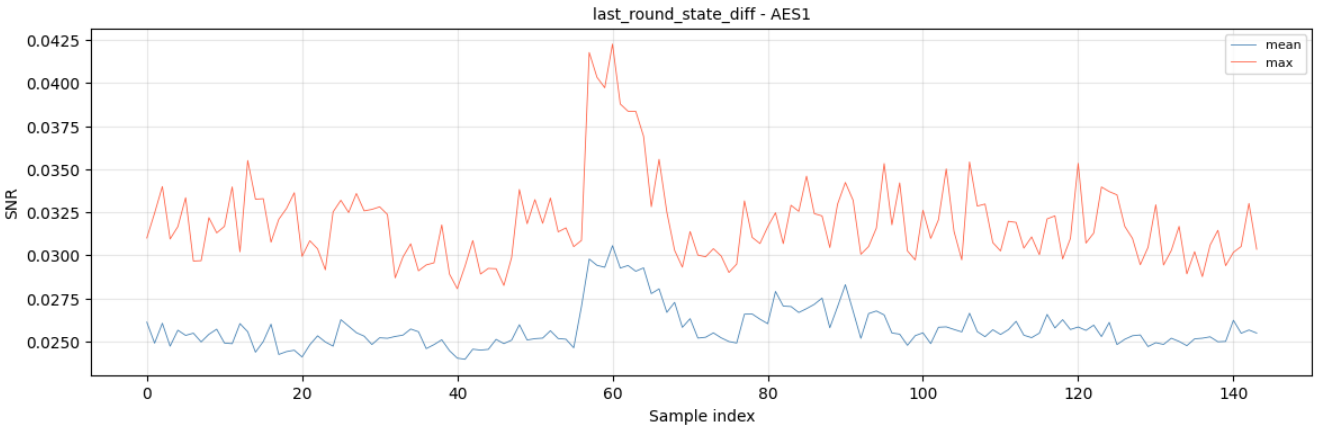

Figure 21: SNR - `last_round_state_diff` - AES1

Similarly, Figure 21, suggests the last AES round's computations to happen around sample 60, corresponding to the 15th clock cycle. These results suggests that round 1-10 lie roughly within clock cycles 5-15, consistent with 1 cycle per AES round for the core encryption operations.

For AES2, an SNR spike for the S-box output is visible at roughly sample index 20. For AES2, the `last_round_state_diff` SNR spike happens roughly at sample index 65, which is slightly later compared to AES1. The higher SNR mean peak for the S-box based models on AES2 suggests that the intermediate explains more of the variance in the traces on AES2 than it does on AES1.

Noteworthy is that, unlike AES1, no strong SNR spike is observed for `round_1_2_state_diff_text`, `round_1_2_state_diff_keymix`, and `custom_1_2_state_diff` on AES2, suggesting that the bit-wise S-box implementation might contribute to these models lack of success in finding correlation on the AES2 traces.

Although the SNR values should not be interpreted as a direct measure of attack success, the results allow a relative comparison between the co-processors and the leakage models under identical measurement conditions. The difference between the measurements indicate that the success of the leakage models may differ between the two AES implementations as their signal strength is nonidentical. Furthermore, the difference between the max and mean SNR values, as observed in the figures presented in Appendix A, indicates a difference of SNR values across the 16 bytes. Some bytes produce high SNR values, while other bytes produce SNR values beneath the mean SNR. This indicates that CPA success might vary across the 16 attacked bytes.

6.2 AES - Correlation Power Analysis

The results for CPA on AES1 (E1) and AES2 (E2) are presented in Table 5 and 6, respectively. The tables show:

1. Recovered: the amount of correct key-guesses (0-16).
2. Corr. (correct): the average correlation coefficient of the correct key guesses, if any. This displays the attack's confidence in the correct key-byte guesses (rounded to four decimals).
3. Corr. (incorrect): the average correlation coefficient of the incorrect key guesses, if any. (rounded to four decimals).
4. Avg PGE (partial guessing entropy): the average rank of the true key byte among all guesses (0-255, lower is better, rounded to one decimal).
5. Avg PGE': the average rank of the true key byte among all guesses, excluding the correct key guesses. Rounded to one decimal.

6.2.1 AES1 (E1)

Model	Rec.	Corr. (correct)	Corr. (inc.)	Avg PGE	Avg PGE'
last_rd_diff	16/16	0.1467	-	0	-
last_rd_diff_alt	4/16	0.1319	0.0424	80.5	107.3
r12_key_mix	4/16	0.1686	0.1512	184.5	246.0
r12_text	3/16	0.0635	0.0410	186.7	229.8
r12_sbox	0/16	-	0.0417	117.8	117.8
sbox_in_out	1/16	0.0454	0.0461	130.6	139.3
sbox_output	1/16	0.0484	0.0513	112.2	119.7
custom_diff	16/16	0.1388	-	0	-

Table 5: CPA results for AES1 (E1).

6.2.2 AES2 (E2)

Model	Rec.	Corr. (correct)	Corr. (inc.)	Avg PGE	Avg PGE'
last_rd_diff	16/16	0.1155	-	0	-
last_rd_diff_alt	4/16	0.1150	0.0441	90.2	120.2
r12_key_mix	2/16	0.0581	0.0453	193.0	220.6
r12_text	0/16	-	0.0316	222.4	222.4
r12_sbox	0/16	-	0.0417	136.4	136.4
sbox_in_out	2/16	0.0532	0.0491	94.1	107.5
sbox_output	0/16	-	0.0468	83.5	83.5
custom_diff	2/16	0.0470	0.0421	77.4	88.4

Table 6: CPA results for AES2 (E2).

Using the last_round_state_diff leakage model and the leakage windows identified in Section 6.1, the full key is recovered in roughly 2K and 3K traces for AES1 and AES2, respectively.

`last_round_state_diff` is the standard leakage model provided by ChipWhisperer that successfully recovered the key, which is consistent with ChipWhisperer's claim that the Hamming distance between the last two rounds is the ideal target for this platform [27]. `custom_1_2_state_diff` recovers the key for AES1 using roughly 1800 traces. However, this model targets intermediate values that depend on multiple input bytes, i.e. the other 15 bytes of the key must be known in order to produce the hypothesized intermediate value. Therefore, the `custom_1_2_state_diff` model is impractical in realistic attack scenarios, but it remains useful for research purposes.

In Figure 22, we plot the results of our trace-count analysis.

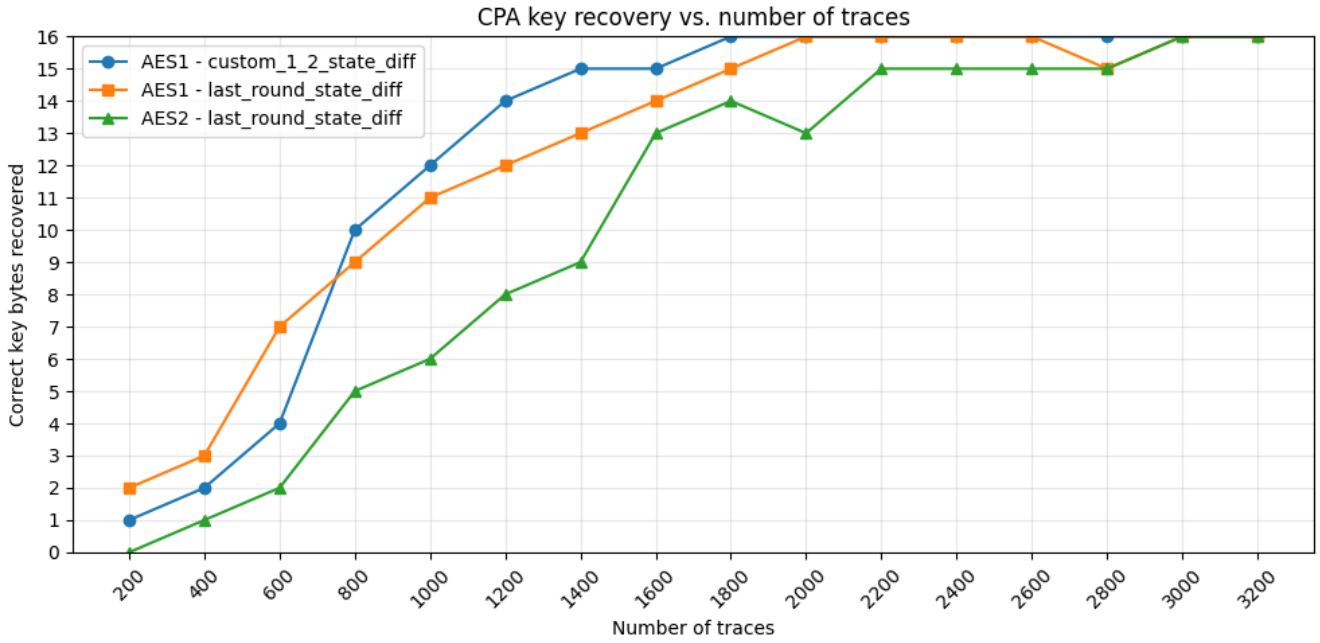

Figure 22: Amounts of traces needed for full key-recovery

We can observe from Figure 22 that the `custom_1_2_state_diff` requires the least amount of traces to recover the full key. `custom_1_2_state_diff` is the only leakage model in which the AES MixColumns operation is performed between the two state values. MixColumns increases diffusion by combining information from multiple state bytes, this may be a factor that contributes to the improved attack performance. For the `last_round_state_diff` model, we observe a higher attack effectiveness for AES1. One possible explanation could be that the bit-wise SubBytes implementation of the AES2 implementation results in more noise compared to the lookup-table based S-box from AES1, attributing to the lower correlation coefficient compared to AES1. Furthermore, from Figure 22, we can observe that the amount of recovered key-bytes decreases by one at one point for both AES1 and AES2. This suggests that at least 1 key-byte has multiple values that show similar correlation and are competing for the best rank at these observed points.

6.3 Validation of the SNR Results

As explained in Section 2, we do not use SNR to decide our CPA attack window. However, by roughly isolating the attack window of the successful leakage models to the observed SNR peaks, we can confirm whether these peaks align with the sample points that were exploited by our CPA attack. Thus, we perform an additional CPA attack on these isolated windows and report the results. The results of the CPA attack for the successful models are presented in Table 7. Note that we omit several columns since the results of these columns are meaningless for successful leakage models.

Model	Attack Window	Rec.	Corr. (correct)
AES1 - last_rd_diff	[50, 70]	16/16	0.1467
AES1 - custom_diff	[20, 40]	16/16	0.1388
AES2 - last_rd_diff	[50, 70]	16/16	0.1155

Table 7: SNR guided CPA attack results.

For all three leakage models we conclude that the leakage points exploited by the CPA attack in Section 6.2 aligns with the SNR peaks observed in Section 6.1. Furthermore, isolating the window resulted in higher correlation coefficients for both the `last_round_state_diff` and `custom_1_2_state_diff` leakage models on AES1. For AES2, the correlation coefficient is identical, meaning that isolating the attack window neither increased or decreased the performance of the `last_round_state_diff` leakage model.

6.4 Conclusions (RQ2)

`last_round_state_diff` is the only model that recovers the full cryptography key for both AES1 and AES2. However, there are several noteworthy additional observations. Despite recovering no key-bytes, `sbox_output` resulted in an average PGE of 83.5/255 (rank 0 = correct key is top guess) on average, which is better than random guessing [8]. This suggests that the leakage model results in above average correlation coefficients for the true key bytes, but the correlation is not strong enough to recover these bytes using 10K traces. Whether more traces increases the success of this model remains open for future research.

On AES1, `last_round_state_diff_alternate`, `round_1_2_state_diff_text` and `round_1_2_state_diff_key_mix` all recovered key-bytes only from the first row of the AES matrix (bytes 0, 4, 8, 12). These positions share the property that ShiftRows leaves them in place. To test whether this could be attributed to model misalignment, several attempts were made using modified versions of these leakage models in which we correct indices for the ShiftRows operation. However, none of these attempts were successful in recovering the rest of the key. Full explanation to this pattern would require hardware-level analysis of the co-processor implementations, which is outside the scope of this thesis.

[8] Random guessing would cause the average PGE to lie around $255/2 = 127.5$.

Moreover, a noteworthy finding is that `round_1_2_state_diff_key_mix`, while recovering four bytes, showed a strong negative correlation on the incorrect key-byte guesses for AES1, with an average PGE' of 246.0, which is significantly worse than random guessing. For AES2, `last_round_state_diff_alternate` and `round_1_2_state_diff_key_mix` also recovered bytes only from the first matrix row. Additionally, `sbox_in_out` recovered 2 key-bytes for AES2. However, the correlation for those bytes was low.

We conclude that `last_round_state_diff` is the only leakage model provided by ChipWhisperer that recovers the full cryptographic key for the AES implementations. The custom leakage model `custom_1_2_state_diff`, which models the state difference between other rounds, can recover the key as well. However, because the hypothesized byte depends on the other 15 input bytes, models like `custom_1_2_state_diff` are not applicable to realistic attack scenarios where the entire key is unknown. Therefore, `custom_1_2_state_diff` remains useful for research purposes only. `custom_1_2_state_diff` did not recover the full key for AES2. However, the average PGE' of 88.4 indicates that key-recovery might be possible with additional traces.

In the case of Ascon and Xoodyak, additional research would be required to assess their leakage exploitability. In Section 2, we explain how these lightweight ciphers absorb the cryptographic key into the state and perform several permutation rounds on this state. These permutation rounds provide wide state diffusion, causing intermediate values to depend on many bits of the key simultaneously. Standard first-order CPA requires predicting an intermediate value from a single guessable key portion, but in the case of Ascon and Xoodyak, these intermediate values depend on many different input bits. In Section 3, we mentioned how Ramezanpour et al. attempted CPA on Ascon, but failed to do so with over 40K traces. Nguyen et al., however, demonstrated that second-order CPA can be used for recovering the full 128-bit Ascon key using a carefully designed selection function targeting the linear diffusion layer output. While statistically significant leakage is confirmed for both implementations, further research is required to determine whether it is exploitable on this specific platform.

7 Discussion & Limitations

In this thesis, we investigated power side-channel leakage of four cryptographic co-processors on the PROACT chip prototype: Ascon, Xoodyak, and two AES implementations. Using Test Vector Leakage Assessment, we concluded that all four co-processors display significant data-dependent leakage, with t-values exceeding the 4.5 threshold. In the case of Ascon and Xoodyak, we observed that the nonce-, AD-, and plaintext-dependent power leakage windows are consistent with their algorithmic flow.

For the AES1 and AES2 co-processors, the SNR analysis was performed to characterize and compare this leakage. We used the `last_round_state_diff` leakage model to recover the full cryptographic key for both AES implementations using only the traces and their corresponding ciphertexts. For the leakage models that failed or only partially succeeded, we observed and reported some noteworthy patterns. Some leakage models recovered bytes only from the first AES matrix row. Attempts were made to adjust the leakage models but they concluded unsuccessful. While custom leakage models like `custom_1_2_state_diff` are not usable in realistic attack scenarios as intermediate values depend on multiple input bytes, we confirmed that, using this model, the PROACT chip

prototype leakage between the first and second round is sufficient for partial key-recovery. For RQ2 we conclude: the observed leakage is exploitable to recover the full cryptographic key for both AES implementations using standard, first-order CPA.

For Xoodyak and Ascon, while statistically significant data-dependent leakage occurs, key-recovery was not attempted. Key recovery using standard CPA would require custom leakage models tailored to the sponge-based construction and permutation-based design of these algorithms, which falls outside the scope of this thesis.

The results of this thesis contribute empirical data to the PROACT project, confirming that the PROACT chip leaks statistically significant data, and that this leakage is exploitable for the AES implementations.

7.1 Limitations

In this thesis, several limitations should be considered when interpreting the results.

First, only the `last_round_state_diff` and `custom_1_2_state_diff` leakage models recovered the full cryptographic key. The other leakage models either failed completely or partially succeeded. For our CPA attempt, only 10000 traces were used, which may have been insufficient for some of the tested models.

In Section 6.2, we show the amount of traces that were required for the models to recover the full encryption key. However, it is important to note that these results are based on a single run. For more reliable trace count statistics, several runs should be performed with random traces. In this thesis, we performed a single run due to the computational intensity of this task.

The experiments in this thesis were performed on traces captured from an unprotected FPGA board in a controlled experimental environment. The results reported in this thesis may differ outside of FPGA implementations or under different measurement conditions. Furthermore, the experimental setup should also be considered, i.e., we captured traces with 4 samples per clock cycle. Different sample rates might influence the success rate of the experiments.

The specific implementations of the algorithms may also contribute to the results. The results discussed in this thesis are specific to the PROACT chip prototype implementations of the algorithms. A different implementation of AES might leak differently even if the algorithm is identical. The claims in this thesis are limited to these specific implementations only and should not be generalized.

Furthermore, in this thesis, only classical CPA was performed. Different techniques such as template or reinforcement-learning based attacks were not explored. These techniques may be able to improve on the classical CPA results reported in this thesis.

Lastly, TVLA confirmed that Ascon and Xoodyak leak statistically significant data-dependent power consumption. However, TVLA results indicate the presence of leakage, but do not imply exploitability. In this thesis, the leakage exploitability of the Ascon and Xoodyak implementations was not explored. We should also consider the drawbacks of TVLA that were mentioned in Section 2. TVLA analysis can be sensitive to false positives and the results should be interpreted carefully. However, the separable leakage windows and their alignment with the algorithmic flows of the algorithms strongly suggest real algorithm-dependent leakage rather than false positives.

8 Future Research

Several potential research opportunities naturally arise from this thesis.

The most obvious question that arises is about the leakage exploitability of the Ascon and Xoodyak co-processors. In this thesis, we confirmed that data-dependent power consumption leakage occurs, but additional research is required to explore whether the leakage is exploitable. To perform a classical CPA attack on these co-processors, leakage models need to be designed that are tailored to the implementation hardware and the permutation-based design of the lightweight ciphers. Alternatively, an attempt could be made to replicate a documented attack like the reinforcement-learning based attack by Ramezanpour et al. [29]. In this paper, the authors demonstrate success of such attack using the same Artix-7 FPGA, making this a natural next step. An attempt could be made to reproduce an attack similar to the one demonstrated by Nguyen et al. [30]. However, in that paper, the authors used a microprocessor running a software implementation, i.e., adjustments would be required to perform it on the PROACT platform.

Additional research could expand on the attacks performed on the AES co-processors. In this thesis, only CPA was attempted. Additional techniques like template attacks and deep-learning based attacks could be explored. Furthermore, a similar experiment could be performed using different experimental setups like a higher sample rate or trace count. Different experimental setups might show whether certain changes to the setup improve the success of the CPA leakage models. Additional research could also be done to explore why certain leakage models were only able to recover key-bytes from the first row of the AES matrix. This pattern could be a result of the specific hardware implementations of the AES algorithm. Future research could link these results to the hardware design of the co-processors.

Lastly, the results of this thesis are specific to the current FPGA-based prototype of the PROACT chip. Once ASIC versions of the chip are produced, re-evaluation might be a natural next step.

References

[1] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *Cryptographic Hardware and Embedded Systems - CHES 2004*, M. Joye and J.-J. Quisquater, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 16–29.

[2] A. Adhikary, A. Basurto, L. Batina, I. Buhan, J. Daemen, S. Mella, N. Mentens, S. Picek, D. L. Ramachandran, A. Sajadi, T. Stefanov, D. Vermoen, and N. Zidaric, "Proact - physical attack resistance of cryptographic algorithms and circuits with reduced time to market," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, I. Skliarova, P. Brox Jiménez, M. Véstias, and P. C. Diniz, Eds. Cham: Springer Nature Switzerland, 2024, pp. 255–266.

[3] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1.2," National Institute of Standards and Technology (NIST), Tech. Rep., 2021. [Online]. Available: https://ascon.iaik.tugraz.at/files/asconv12-nist.pdf

[4] M. S. Turan, "Ascon-based lightweight cryptography standards for constrained devices," National Institute of Standards and Technology (U.S.), Tech. Rep., 2025. [Online]. Available: https://api.semanticscholar.org/CorpusID:280273255

[5] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 01 2002.

[6] National Institute of Standards and Technology, "Advanced encryption standard (aes)," National Institute of Standards and Technology (U.S.), Tech. Rep. FIPS 197, 2023. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf

[7] CAESAR Committee, "CAESAR: Competition for authenticated encryption: Final portfolio," 2019. [Online]. Available: https://competitions.cr.yp.to/caesar-submissions.html

[8] F. Mendel, "The ascon family: Lightweight authenticated encryption, hashing, and more," Presentation for NIST, 2023. [Online]. Available: https://csrc.nist.gov/csrc/media/Presentations/2023/the-ascon-family/images-media/june-21-mendel-the-ascon-family.pdf

[9] M. S. Turan, K. McKay, D. Chang, J. Kang, and J. Kelsey, "Ascon-based lightweight cryptography standards for constrained devices," in *NIST Special Publication (SP) NIST SP 800–232 ipd*. National Institute of Standards and Technology, 2024.

[10] J. Daemen, S. Hoffert, M. Peeters, G. Assche, and R. Keer, "Xoodyak, a lightweight cryptographic scheme," *IACR Transactions on Symmetric Cryptology*, pp. 60–87, 06 2020.

[11] J. Daemen, S. Hoffert, M. Peeters, G. V. Assche, and R. V. Keer, "Xoodyak, a lightweight cryptographic scheme," National Institute of Standards and Technology, Tech. Rep., 2021. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/xoodyak-spec-final.pdf

[12] O. S. Fadl, M. F. Abu-Elyazeed, M. B. Abdelhalim, H. H. Amer, and A. H. Madian, "Accurate dynamic power estimation for cmos combinational logic circuits with real gate delay model," *Journal of Advanced Research*, vol. 7, no. 1, pp. 89–94, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2090123215000296

[13] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology — CRYPTO' 99*, M. Wiener, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397.

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

[15] NewAE Technology Inc., "ChipWhisperer wiki," 2023, accessed: 2026. [Online]. Available: https://chipwhisperer.readthedocs.io/en/latest/

[16] ——, "CW305 artix FPGA target," accessed: 2026. [Online]. Available: https://chipwhisperer.readthedocs.io/en/latest/Targets/CW305%20Artix%20FPGA.html

[17] ——, "ChipWhisperer-Husky documentation," accessed: 2026. [Online]. Available: https://chipwhisperer.readthedocs.io/en/latest/Capture/ChipWhisperer-Husky.html

[18] ——, "ChipWhisperer software," accessed: 2026. [Online]. Available: https://github.com/newaetech/chipwhisperer

[19] ——, "Chipwhisperer scope api documentation," accessed: 2026. [Online]. Available: https://chipwhisperer.readthedocs.io/en/latest/scope-api.html

[20] ——, "Chipwhisperer target api documentation," accessed: 2026. [Online]. Available: https://chipwhisperer.readthedocs.io/en/latest/target-api.html

[21] ——, "Chipwhisperer capture api documentation," accessed: 2026. [Online]. Available: https://chipwhisperer.readthedocs.io/en/latest/capture-api.html

[22] ——, "Chipwhisperer analyzer api documentation," accessed: 2026. [Online]. Available: https://chipwhisperer.readthedocs.io/en/latest/analyzer-api.html

[23] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, "A testing methodology for side-channel resistance validation," in *Non-Invasive Attack Testing Workshop (NIAT)*, 2011. [Online]. Available: https://csrc.nist.gov/csrc/media/events/non-invasive-attack-testing-workshop/documents/08_goodwill.pdf

[24] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer US, 2007. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-38162-6

[25] Y. Wang and M. Tang, "A survey of side-channel leakage assessment," *Electronics*, vol. 12, no. 16, 2023. [Online]. Available: https://www.mdpi.com/2079-9292/12/16/3461

[26] F.-X. Standaert, E. Peeters, G. Rouvroy, and J.-J. Quisquater, "An overview of power analysis attacks against field programmable gate arrays," *Proceedings of the IEEE*, vol. 94, pp. 383 – 394, 03 2006.

[27] A. Dewar, J.-P. Thibault, and C. O'Flynn, "Naean0010: Power analysis on fpga implementation of aes using cw305 & chipwhisperer r o," 2020. [Online]. Available: https://media.newae.com/appnotes/NAE0010_Whitepaper_CW305_AES_SCA_Attack.pdf

[28] K. Mohajerani, L. Beckwith, A. Abdulgadir, J.-P. Kaps, and K. Gaj, "Lightweight champions of the world: Side-channel resistant open hardware for finalists in the nist lightweight cryptography standardization process," *ACM Trans. Embed. Comput. Syst.*, vol. 24, no. 5, Sep. 2025. [Online]. Available: https://doi.org/10.1145/3677320

[29] K. Ramezanpour, A. Abdulgadir, W. Diehl, J.-P. Kaps, and P. Ampadu, "Active and passive side-channel key recovery attacks on ascon," in *Proc. NIST Lightweight Cryptogr. Workshop*, 2020, pp. 1–27.

[30] V. S. Nguyen, V. Grosso, and P.-L. Cayrel, "Practical second-order cpa attack on ascon with proper selection function," in *Constructive Approaches for Security Analysis and Design of Embedded Systems*, M. Rivain and P. Sasdrich, Eds. Cham: Springer Nature Switzerland, 2026, pp. 311–339.

[31] PyCryptodome Developers, "PyCryptodome," accessed: 2026. [Online]. Available: https://www.pycryptodome.org

[32] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright *et al.*, "Scipy 1.0: fundamental algorithms for scientific computing in python," *Nature methods*, vol. 17, no. 3, pp. 261–272, 2020.

[33] C. Harris, K. Millman, S. Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. Smith, R. Kern, M. Picus, S. Hoyer, M. Kerkwijk, M. Brett, A. Haldane, J. Río, M. Wiebe, P. Peterson, and T. Oliphant, "Array programming with numpy," *Nature*, vol. 585, pp. 357–362, 09 2020.

[34] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

A SNR Analysis Plots

A.1 AES1

1. `last_round_state_diff`

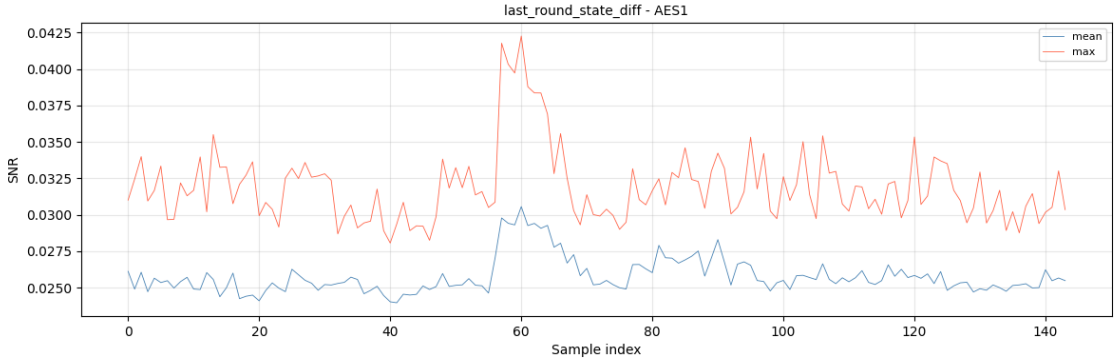

Figure 23: SNR - `last_round_state_diff` - AES1

In Figure 23, we observe a clear spike in SNR roughly between samples 50-70.

2. `last_round_state_diff_alternate`

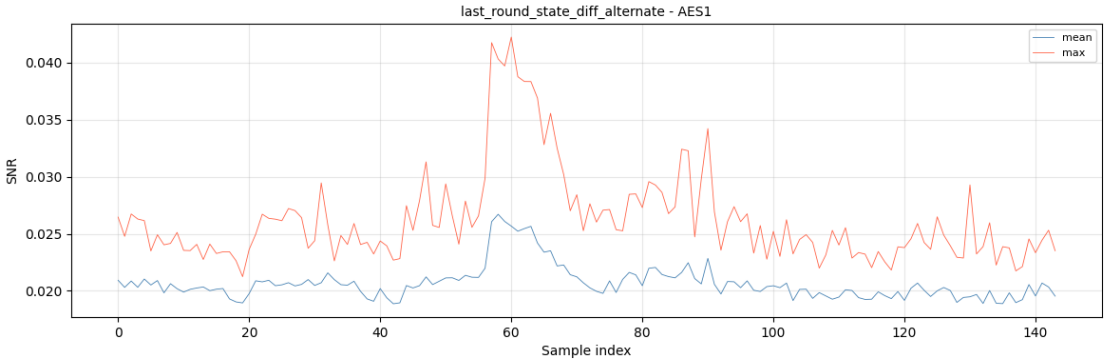

Figure 24: SNR - `last_round_state_diff_alternate` - AES1

In Figure 24, we observe a spike roughly between samples 50-70.

3. `sbox_output`

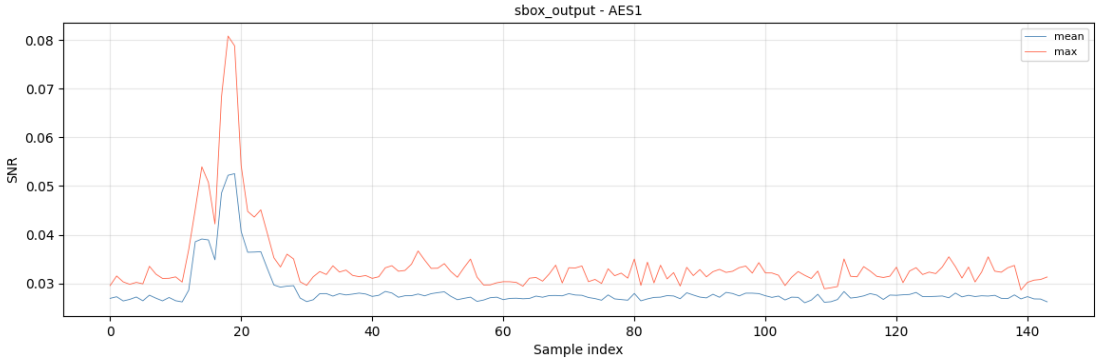

Figure 25: SNR - `sbox_output` - AES1

In Figure 25, we observe a high SNR spike roughly between samples 10-25.

4. `sbox_in_out_diff`

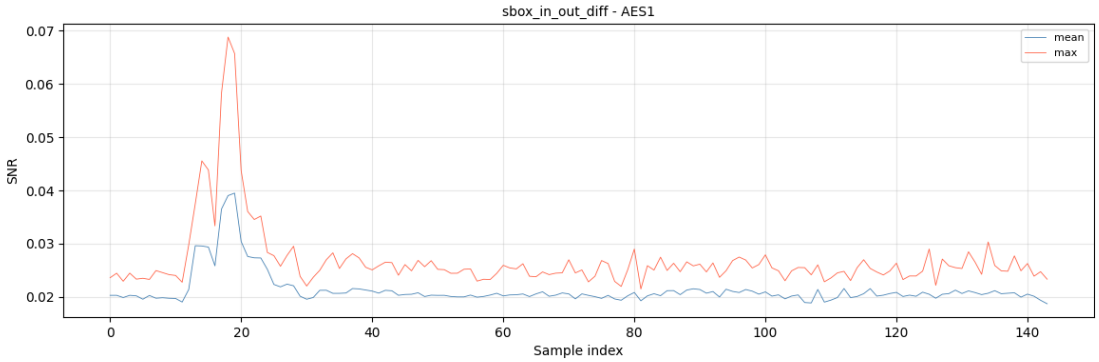

Figure 26: SNR - `sbox_in_out_diff` - AES1

In Figure 26, we observe a high SNR spike roughly between samples 10-25.

5. `round_1_2_state_diff_text`

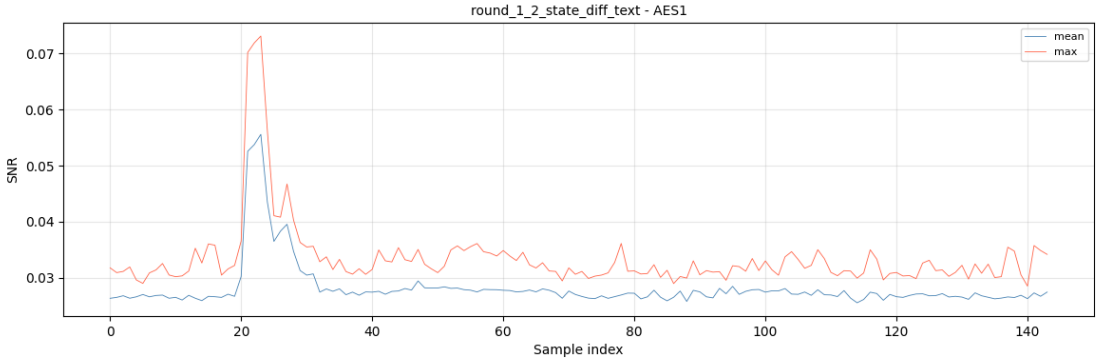

Figure 27: SNR - `round_1_2_state_diff_text` - AES1

In Figure 27, we observe an SNR spike roughly between samples 20-30.

6. `round_1_2_state_diff_keymix`

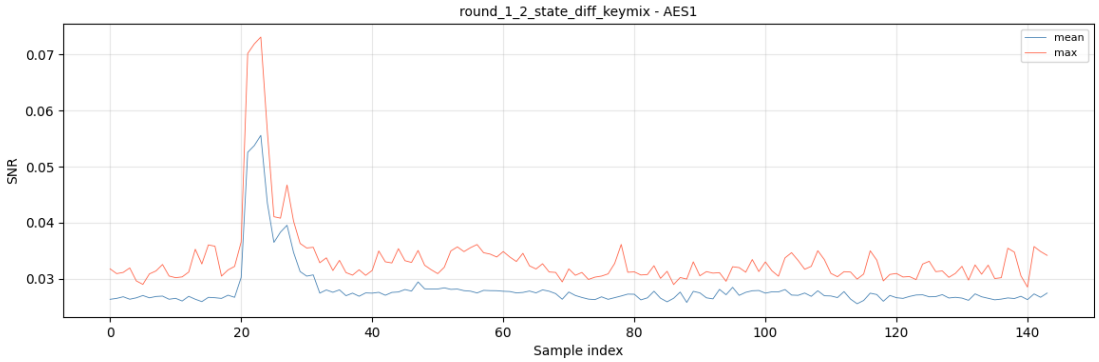

Figure 28: SNR - `round_1_2_state_diff_keymix` - AES1

In Figure 28, we observe an SNR spike roughly between samples 20-30.

7. `round_1_2_state_diff_sbox`

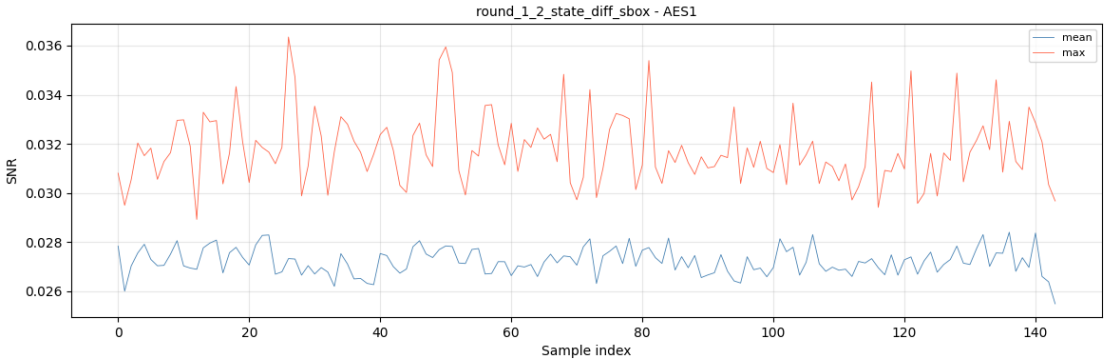

Figure 29: SNR - `round_1_2_state_diff_sbox` - AES1

In Figure 29, we observe no significant SNR spike.

8. `custom_1_2_state_diff`

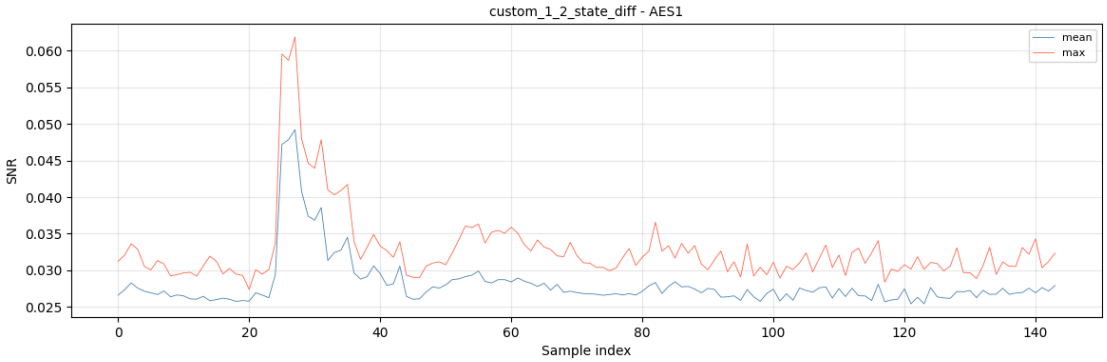

Figure 30: SNR - `custom_1_2_state_diff` - AES1

In Figure 30, we observe a significant SNR spike roughly between samples 20-40.

A.2 AES2

(a) `last_round_state_diff`

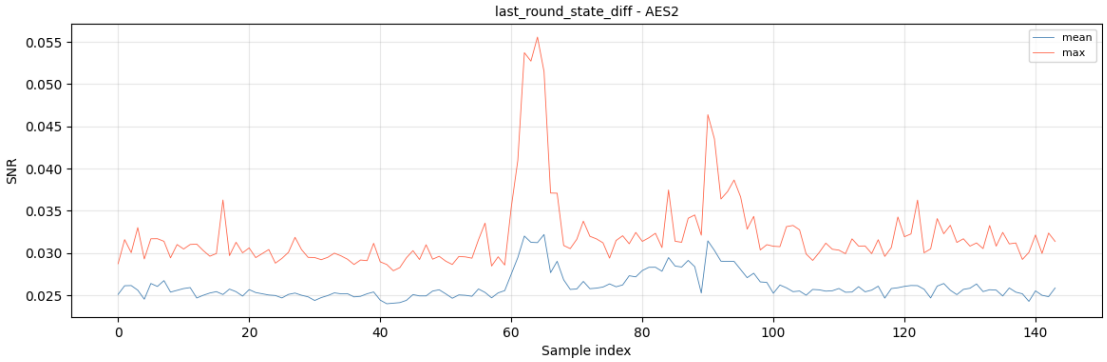

Figure 31: SNR - `last_round_state_diff` - AES2

In Figure 31, we observe a spike roughly between samples 55-70.

(b) `last_round_state_diff_alternate`

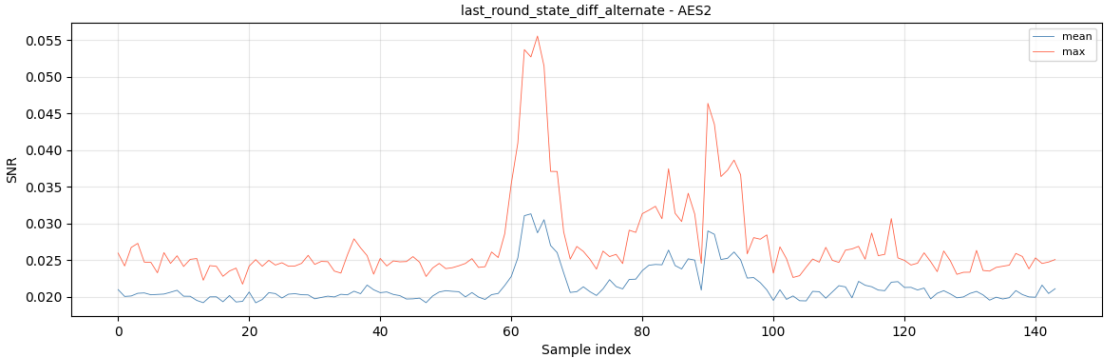

Figure 32: SNR - `last_round_state_diff_alternate` - AES2

In Figure 32, we observe a spike roughly between samples 55-70.

(c) `sbox_output`

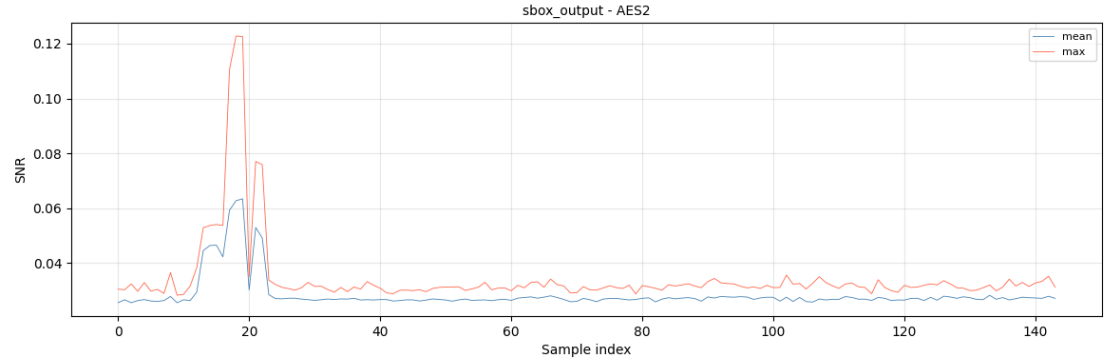

Figure 33: SNR - `sbox_output` - AES2

In Figure 33, we observe a high SNR spike roughly between samples 10-25.

(d) `sbox_in_out_diff`

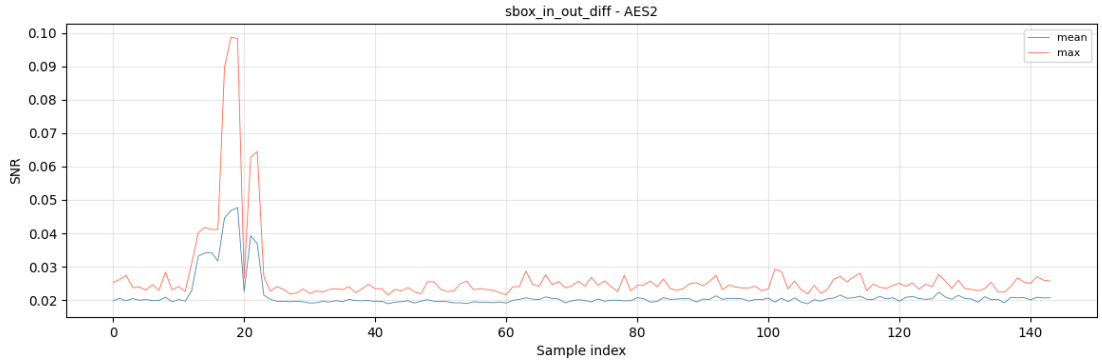

Figure 34: SNR - `sbox_in_out_diff` - AES2

In Figure 34, we observe a high SNR spike roughly between samples 10-25.

(e) `round_1_2_state_diff_text`

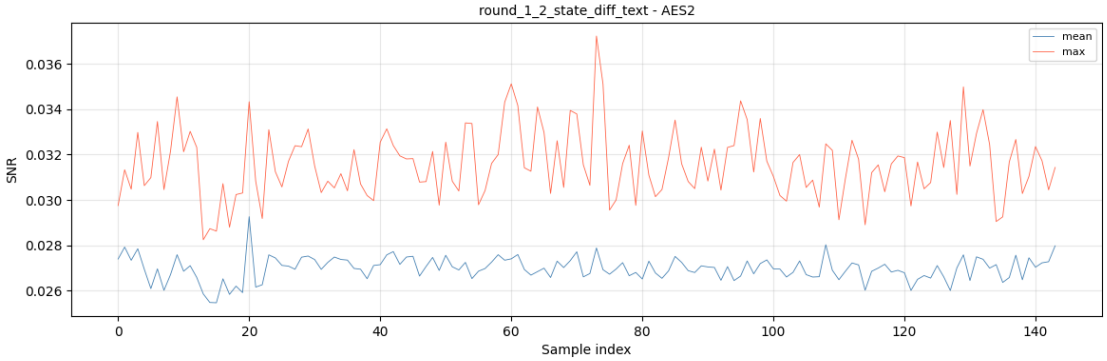

Figure 35: SNR - `round_1_2_state_diff_text` - AES2

In Figure 35, we observe no significant SNR peak.

(f) `round_1_2_state_diff_keymix`

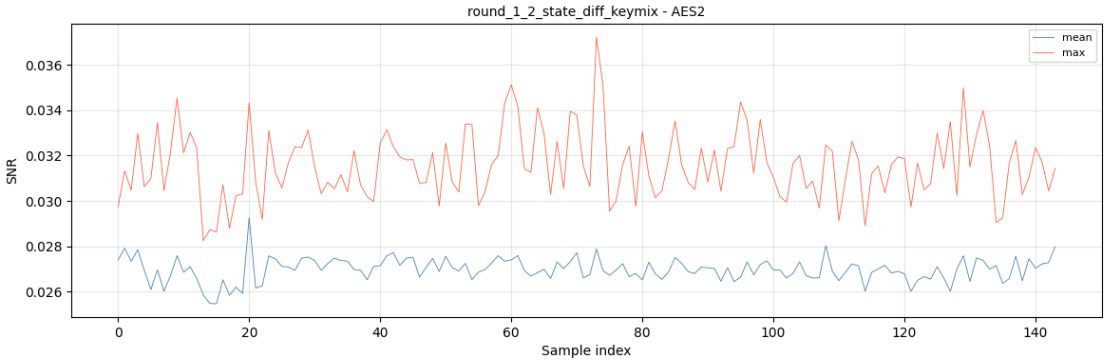

Figure 36: SNR - `round_1_2_state_diff_keymix` - AES2

In Figure 36, we observe no significant SNR peak.

(g) `round_1_2_state_diff_sbox`

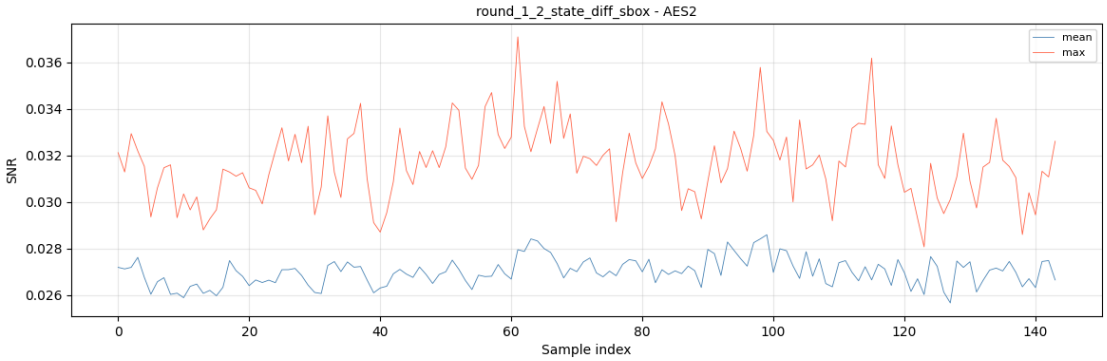

Figure 37: SNR - `round_1_2_state_diff_sbox` - AES2

In Figure 37, we observe no significant SNR peak.

(h) `custom_1_2_state_diff`

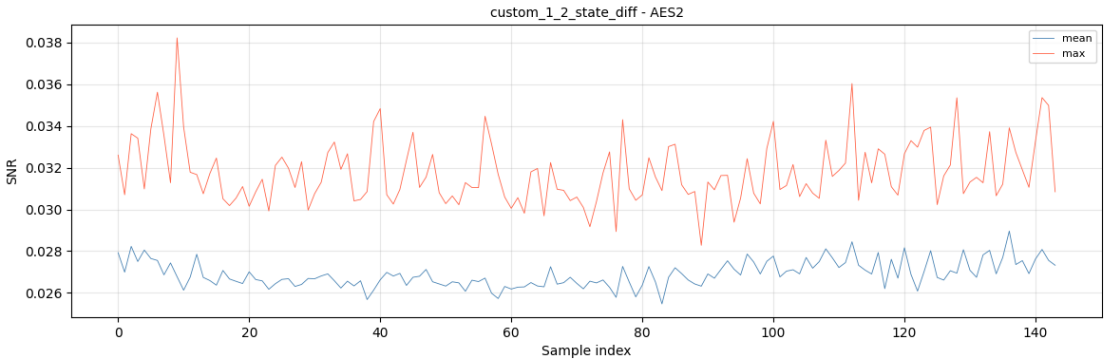

Figure 38: SNR - `custom_1_2_state_diff` - AES2

In Figure 38, we observe no significant SNR peak. We can observe a slight peak roughly between 0-15. However, comparing this to the results of the other models leads us to the conclusion that this is noise (meaningful variance of this intermediate should not be possible at this stage).