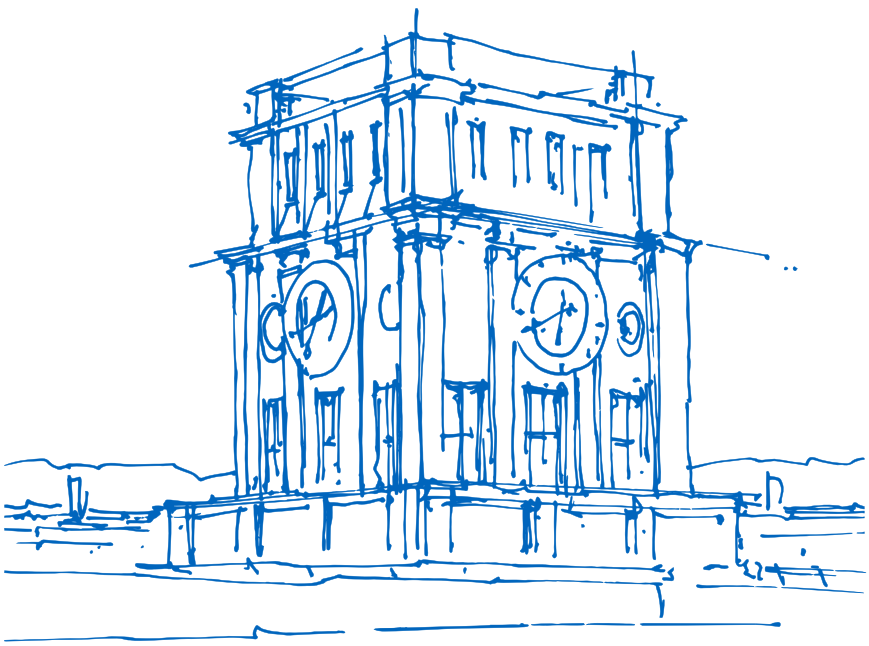
Chair for Security in Information Technology
TUM School of Computation, Information and Technology
Technical University of Munich

Side Channel Analysis of CROSS

Template Attack on a Code-Based PQC Signature Algorithm

Stephan Schwarzmann



Chair for Security in Information Technology
TUM School of Computation, Information and Technology
Technical University of Munich

Side Channel Analysis of CROSS

Template Attack on a Code-Based PQC Signature Algorithm

Stephan Schwarzmann

Thesis for the attainment of the academic degree

Master of Science (M.Sc.)

at the TUM School of Computation, Information and Technology of the Technical University of Munich.

Examiner:
Prof. Dr.-Ing. Georg Sigl

Supervisor:
Jonas Schupp

Submitted:
Munich, 10.06.2025

Abstract

The expected advent of quantum computers that are strong enough to break current asymmetric encryption schemes triggered the development of PQC, which refers to cryptographic schemes that are thought to be resistant against quantum computers. After NIST standardized the first PQC algorithms as a result of a *call for proposals*, the diversity of mathematical problems used for signature algorithms turned out to be limited: all general purpose signature algorithms are based on structured lattices. Since no proof exists that such schemes are secure, the security relies on the assumption that they cannot be broken with state-of-the-art and even future knowledge. The limited experience with the new mathematical approaches used for PQC makes it reasonable to develop different schemes in case one or more of the underlying designs can be broken in the future. Thus, NIST requested more signature algorithms, mainly ones that are not based on lattices. CROSS is a such signature algorithm based on *codes*. In this work, we attempt to retrieve the secret key used in CROSS by performing a template attack. Those attacks are a variant of SCA. We find that the vector-matrix multiplication is vulnerable to side-channel analysis; critical values related to the key are processed repeatedly with known public-key values, and small values can be isolated for building hypotheses with a size that can be tested with reasonable computational effort. Using a Hamming weight leakage model, measurements with a simple setup reveal strong correlations between those values and correct hypotheses in a limited hypothesis space. This allows for the recovery of the key with a reasonable amount of effort, at least for some configurations of the CROSS algorithm.

Acknowledgements

This thesis benefited from AI-based writing tools, which were used for phrasing suggestions, language polishing, and generating illustrative figures under the author's supervision and review.

I would like to express my sincere gratitude to my supervisor for his valuable guidance and support throughout this work. In particular, his suggestions regarding the vector-matrix operation $\mathbf{s} = \mathbf{u}\mathbf{H}^\top$ and the reduction of the hypothesis space via the z-value structure provided crucial insights that significantly influenced the development of the attack strategy.

Contents

1 Introduction

1.1 Background on Post-Quantum Cryptography

Once quantum computers with a sufficient number of logical qubits become available, they will enable attacks against several primitives that are the backbone of currently used cryptography. Grover's algorithm [8] reduces the effective security level of symmetric cryptography schemes to approximately half the key size. For example, AES-256 under quantum attack offers security comparable to AES-128 against classical attacks. A similar reduction applies to hash functions. These security reductions can be countered by increased key sizes.

However, the threat to asymmetric cryptographic schemes is significantly more severe. Shor's algorithm [13] allows efficient factorization and discrete logarithm computations, rendering current asymmetric algorithms such as RSA and elliptic curve cryptography (ECC) insecure in the presence of large-scale quantum computers.

Even though such machines may still be years or decades away, the confidentiality of long-term sensitive data is already at risk due to the *harvest now, decrypt later* strategy: adversaries can store encrypted data today and decrypt it once quantum capabilities become available. This anticipated risk has accelerated the push toward post-quantum cryptography (PQC). In 2016, the National Institute of Standards and Technology (NIST) initiated a call for proposals to standardize quantum-resistant key encapsulation mechanisms (KEMs) and digital signature schemes [4]. Since then, several PQC algorithms have been selected for standardization. The finalized signature schemes include CRYSTALS-Dilithium [14], a lattice-based scheme, and SPHINCS+ [15], which is based on hash functions. FALCON, another lattice-based scheme, has also been selected but is still undergoing finalization at of April 2025 [7].

Notably, these algorithms rely on only two classes of underlying mathematical problems. To diversify the post-quantum portfolio, NIST issued a second call in 2022 for signature schemes based on different mathematical foundations [5]. One of the resulting submissions is the Codes and Restricted Objects Signature Scheme (CROSS), the signature scheme investigated in this thesis.

1.2 Introduction to Side-Channel Analysis

Ever since Paul Kocher demonstrated the practicality of side-channel attacks [10, 11], cryptographic algorithms not only need to be secure against cryptanalysis of the mathematical structure, but also need to withstand side-channel attacks (SCAs).

If an attacker has access to a device processing a secret key, the device must resist such analysis targeted at the implementation of the cryptographic scheme. For the development of new cryptographic algorithms, vulnerability against SCAs is an additional aspect that needs to be considered when evaluating its security. Some algorithms are naturally more prone to such attacks than others, but typically, specific countermeasures have to be implemented to secure algorithms. Thus, developers need to learn about side-channel vulnerabilities to understand which countermeasures are needed to either thwart attacks or at least mitigate the risks by increasing the effort required for a successful attack.

SCAs exploit implementation details of cryptographic algorithms rather than utilizing weaknesses in their mathematical design. Such attacks can use leakage of information in different physical domains. The most common leakage sources are execution timings, shared cache accesses, electromagnetic emanation and power consumption.

Power analysis, in particular, is widely used due to its low equipment requirements, ease of setup, and practical effectiveness. Unlike electromagnetic analysis (EMA), it assumes a weaker attacker model, only requiring access to the device's power consumption rather than close physical proximity

for electromagnetic (EM) measurements. These properties make it a good initial approach to examine a new cryptographic implementation.

The most straightforward power analysis technique is simple power analysis (SPA), which refers to visual inspection of power measurements (traces) to differentiate the operations performed on a device. Differential power analysis (DPA) employs statistical methods to eliminate noise and find correlations across many traces from a device under attack. Template attacks, on the other hand, initially require a large number of traces collected from a separate profiling device that is under the attacker's control. The resulting model can then be applied to attack similar target devices and may only need a few traces—or even a single one—to recover secret information.

1.3 Overview of the CROSS Signature Algorithm

CROSS is a PQC signature algorithm submitted for the call for additional digital signature proposals [5]. Researchers from the Technical University of Munich (TUM) contributed to this scheme, which makes it an interesting subject for investigation at TUM. CROSS combines a non-interactive form of zero-knowledge proofs (ZKPs) with a variant of the syndrome decoding problem (SDP), and thus falls into the category of signature schemes whose security relies on codes.

The zero-knowledge scheme in CROSS is based on a 5-pass protocol (CROSS-ID), in which a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ exchange five messages as illustrated in Figure 1.1: A *commitment* from $\mathcal{P}$, a *challenge* from $\mathcal{V}$ and a corresponding *response* from $\mathcal{P}$, and another set of *challenge* and *response*. CROSS-ID is a *q2*-Identification scheme, i.e., the first and second challenge can take $|\beta| = q$ and $|b| = 2$ values, respectively.

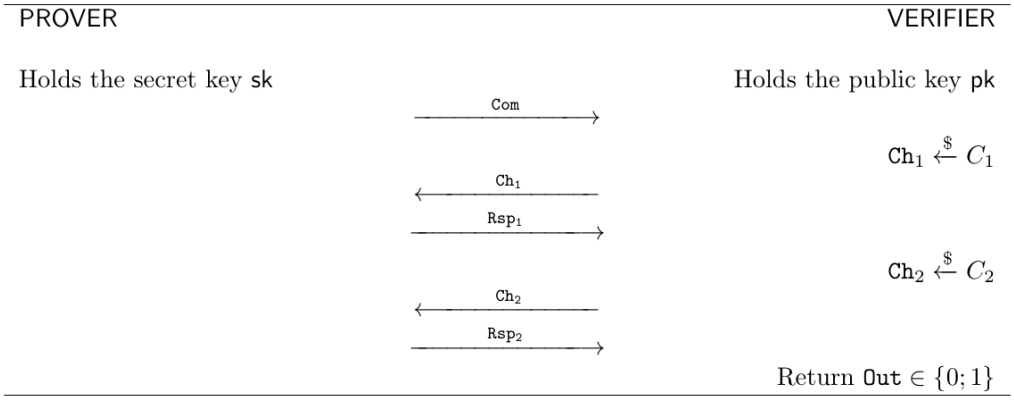


Figure 1.1 5-pass protocol

ZKPs are probabilistic. Using such a 5-pass protocol once, a malicious $\mathcal{P}$ can prepare the correct answer for one of the two possible values of the second challenge ($b = 0$ or $b = 1$) and has thus a certain chance of providing the correct answer to $\mathcal{V}$ without knowing the secret. In the case of $q2$, where the second challenge can take one of two possible values, this chance is roughly 50%. This is the so-called soundness error ε for one execution of the protocol. ZKPs execute protocols repeatedly to reduce the total soundness error. In CROSS those repetitions are called *rounds*. With t rounds, the total soundness error is ε^t. The chance for a malicious prover to mislead a verifier decreases with the number of rounds.

While ZKPs are interactive, they can be transformed into non-interactive schemes using the *Fiat-Shamir transform* [6], an approach that is already employed by several digital signature schemes. The non-interactive version of ZKP forces $\mathcal{P}$ to commit to values that reflect which of the two possible second challenges he can answer correctly and which not. Since the challenges depend on the commitment values, a malicious actor impersonating $\mathcal{P}$ has to decide on which second challenge to answer ($b = 0$ or $b = 1$) before knowing the actual challenge. The challenges are derived using a pseudo-random hash

function with the commitment as part of the input, so the prover cannot manipulate the challenges in a way that he can always prepare for the challenge and generate correct answers.

The ZKP used in CROSS is based on *linear codes* and employs the hardness of decoding a *restricted error vector*, i.e., the values of the error vector are in a subgroup $\mathbb{E}$ of the field $\mathbb{F}$ used for the linear code. Both decoding problems used in CROSS, the restricted syndrome decoding problem (R-SDP) and the R-SDP in a Subgroup (R-SDP(G)), are proven to be NP-hard. These problems form the mathematical basis for the scheme's security guarantees.

Security Assumptions. The security of CROSS relies on several well-established cryptographic principles. At its core, it uses an interactive zero-knowledge protocol, which is transformed into a non-interactive signature scheme using the Fiat-Shamir heuristic—a widely trusted design approach with strong theoretical foundations. The scheme also depends on standard assumptions: the hardness of decoding structured linear codes and the collision resistance of cryptographic hash functions. Together, these components yield a signature scheme built from thoroughly studied, broadly accepted cryptographic building blocks.

Configurations. CROSS supports several configurations to suit different requirements:

- **Algorithms**: R-SDP and R-SDP(G)
- **Security Category**: 1, 3, 5
- **Optimization Corner**: *fast*, *balanced*, *small*

R-SDP uses a subgroup $\mathbb{E} = \langle g \rangle \subseteq \mathbb{F}_p^*$ for the error vector, whereas R-SDP(G) uses a smaller subgroup $G \subset \mathbb{E}^n$, with $|G| < |\mathbb{E}^n| = z^n$. The security categories refer to NIST security levels and are difficulty equivalents for AES-128 (category 1), AES-192 (3) and AES-256 (5). The optimizations *fast*, *balanced* and *small* determine the trade-off between signature size and execution speed, e.g., *fast* for fast executions with larger signatures, or vice-versa, *small* for shorter signatures with higher computational effort.

Table 1.1 is taken from the CROSS specification and presents the configurations along with their influence on key and signature sizes.

1.4 Related Work

Template attacks, first introduced in 2002 [3], form the basis for our side-channel strategy. These attacks are well established and serve as a foundational technique in profiling-based power analysis.

To support our implementation, we rely on the side-channel analysis library (SCALib) [2], a software library that provides tooling for trace analysis, signal processing, and profiling model generation.

To the best of our knowledge, no published side-channel attacks against CROSS existed at the beginning of this thesis. However, during the course of this work, researchers at TU Munich independently demonstrated a successful DPA attack on CROSS [12].

1.5 Contributions

We present a template attack on CROSS-R-SDP-fast-1, targeting the signature generation phase via power side-channel analysis. Our results show that full key recovery is achievable using 5,000 profiling traces and, in some cases, with only a single signature trace, all captured with standard equipment under realistic leakage conditions. This enables the generation of valid signatures by the attacker. However, success is not guaranteed for arbitrary traces due to variability in intermediate values and leakage quality.

We analyze the confidence margins between the best and second-best hypotheses and show that relative differences are frequently too small to support reliable threshold-based filtering.

Additionally, we provide insights into specific properties of the CROSS algorithm and their implications for attacking the scheme via SCAs, as well as suggestions to improve the security against

Algorithm and Security Category	Optim. Corner	p	z	n	k	m	t	w	Pri. Key Size (B)	Pub. Key Size (B)	Signature Size (B)
CROSS-R-SDP **1**	**fast**	127	7	127	76	-	163	85	16	61	19152
	balanced	127	7	127	76	-	252	212	16	61	12912
	small	127	7	127	76	-	960	938	16	61	10080
CROSS-R-SDP **3**	**fast**	127	7	187	111	-	245	127	24	91	42682
	balanced	127	7	187	111	-	398	340	24	91	28222
	small	127	7	187	111	-	945	907	24	91	23642
CROSS-R-SDP **5**	**fast**	127	7	251	150	-	327	169	32	121	76298
	balanced	127	7	251	150	-	507	427	32	121	51056
	small	127	7	251	150	-	968	912	32	121	43592
CROSS-R-SDP(G) **1**	**fast**	509	127	55	36	25	153	79	16	38	12472
	balanced	509	127	55	36	25	243	206	16	38	9236
	small	509	127	55	36	25	871	850	16	38	7956
CROSS-R-SDP(G) **3**	**fast**	509	127	79	48	40	230	123	24	59	27404
	balanced	509	127	79	48	40	255	176	24	59	23380
	small	509	127	79	48	40	949	914	24	59	18188
CROSS-R-SDP(G) **5**	**fast**	509	127	106	69	48	306	157	32	74	48938
	balanced	509	127	106	69	48	356	257	32	74	40134
	small	509	127	106	69	48	996	945	32	74	32742

Table 1.1 Parameter choices, keypair and signature sizes recommended for both CROSS-R-SDP and CROSS-R-SDP(G), assuming NIST security categories 1, 3, and 5, respectively.

side-channel analysis. Additionally, we explore how specific properties of the CROSS algorithm affect side-channel leakage and attack viability. These observations provide both practical insights for improving SCA resistance and a foundation for further research into alternative CROSS configurations.

1.6 Outline

This thesis proceeds as follows: In chapter 2, we define the relevance and objectives of the work. chapter 3 presents an analysis of the target algorithm and details the steps taken to perform the attack. The experimental results are provided in chapter 4, followed by a discussion in chapter 5. Finally, we summarize the findings and conclude the thesis in chapter 6.

2 Research Question and Objectives

2.1 Problem Statement and Research Objectives

The CROSS algorithm was submitted in response to NIST's call for additional digital signature schemes and, at the beginning of this thesis, is available in version v1.2. As a candidate for standardization, it is currently under review by the PQC community. To date, no public analysis of the algorithm's resistance against SCAs exists. Evaluating the side-channel resilience of CROSS is therefore a necessary step in assessing its suitability for deployment.

This thesis investigates the vulnerability of CROSS to side-channel attacks by performing a template attack based on power consumption measurements. The goal is to recover the full secret key from a device under test (DUT) executing the CROSS algorithm and to identify implementation-level weaknesses that make such attacks feasible. Our focus is on the signing operation, as this is the most likely target in a real-world attack scenario.

Rather than aiming to exploit the scheme, our objective is to provide insights into its side-channel leakage characteristics to assist developers and researchers in securing CROSS against such attacks.

2.2 Scope and Limitations

Throughout this thesis, references to the CROSS specification pertain to version 1.2, unless explicitly noted otherwise. The specification and reference implementation are publicly available [1]. As of version 1.2, CROSS includes no explicit protections against SCAs.

This work focuses on evaluating the vulnerability of CROSS to side-channel attacks, specifically targeting the signing operation via power analysis. We perform a template attack, described in detail in section 3.2, which relies on building a leakage model from a set of profiling traces and then recovering the secret key using only a few attack traces—potentially as few as one.

Our experiments use the following setup:

- a ChipWhisperer CW308 UFO board to host the target,
- a ChipWhisperer target board with an STM32F4 microcontroller (ARM Cortex-M4) as the DUT,
- and a Picoscope 6402C digital storage oscilloscope (DSO) for power trace acquisition.

We select the *CROSS-R-SDP-1-fast* parameter set for our initial experiments. This configuration corresponds to the lowest security level. Due to its shorter secret key and smaller underlying field ($p = 127$), it presents a reduced attack complexity.

We assume access to the reference implementation, enabling us to:

- choose the message and seed used during signing on the DUT,
- run modified code to isolate specific rounds for profiling,
- and collect power traces from the device during execution.

In the attack phase, we assume a realistic adversary who does not control the message or seed but uses the leakage model obtained during profiling.

Trace alignment is simplified in this work by using the same trace acquisition process for both profiling and attack phases. We do not explore the challenges of trace misalignment, which would arise in a real-world attack scenario. Ensuring accurate alignment is still necessary, but does not require additional signal processing beyond consistent measurement procedures.

Limitations of this study are discussed further in section 3.4.

3 Attack Strategy and Implementation

3.1 Preliminaries

3.1.1 Terminology and Notation

We use the following conventions for notation throughout this thesis:

- Bold Latin letters are used for vectors and matrices in field space, with lowercase (e.g., $\mathbf{v}$) denoting vectors and uppercase (e.g., $\mathbf{H}$) for matrices.
- Greek letters such as η and σ are used to represent exponent-space vectors and isometries.
- Scalars appear in standard math italic font, e.g., a, b, x.
- C source code expressions are written in monospace font, e.g., `c_var`.
- Number sets are denoted using double-struck fonts, e.g., $\mathbb{F}_p$ for a finite field of order p, or $\mathbb{Z}$ for integers.
- $\mathbb{F}_p^n$ denotes a vector space of length n over a finite field with p elements.
- $\mathbf{H}^\top$ denotes the transpose of a matrix $\mathbf{H}$.
- The notation $[\mathbf{H}]_{i,j}$ denotes the element in row i and column j of matrix $\mathbf{H}$.
- The operator $\star$ denotes the component-wise (Hadamard) product. Its semantics depend on the domain: in $\mathbb{F}_p$, it represents multiplication; in $\mathbb{F}_z$, it denotes addition modulo z.

Vector Slicing Notation. Given a vector $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$, we denote by $\mathbf{v}[j{:}k]$ the subvector containing the elements $v_j, v_{j+1}, \ldots, v_k$. This notation is *inclusive* on both endpoints and assumes *0-based indexing*, i.e., $\mathbf{v}[0{:}2] = (v_0, v_1, v_2)$. Such slices are commonly used to extract segments of vectors (e.g., $\mathbf{u}[0{:}K-1]$) corresponding to a specific subset of components.

3.1.2 Variable Mapping between Specification and Code

We adopt the designators as introduced in the CROSS specification and the codebase, which use different notational conventions are Table 3.1 provides a clear mapping between these notations, along with their algebraic context and intended meaning.

The CROSS specification defines algebraic structures used for sampling and encoding:

- The cyclic group $\mathbb{E} = \langle g \rangle \subset \mathbb{F}_p^*$ is the multiplicative subgroup of nonzero field elements generated by g.
- A structured subgroup $G \subset \mathbb{E}^n$ with $|G| < |\mathbb{E}^n| = z^n$ is used to constrain the error vector in the R-SDP problem.

Concept	Spec Notation	Code Notation	Space
Error vector (structured)	$\mathbf{e}$	`e`	$\mathbb{E}^n \subset \mathbb{F}_p^n$
Transformed error vector	$\mathbf{e}' = \sigma \star \mathbf{e}$	`e_tilde`	$\mathbb{E}^n$
Isometry exponent vector	σ	`sigma`	$\mathbb{F}_z^n$
Converted isometry (powers of 2)	$\mathbf{v}$	`v`	$\mathbb{F}_p^n$
Error vector in exponent form	η	`eta`	$\mathbb{F}_z^n$
Transformed error vector $\mathbf{e}'$, exponent form	$\eta' = \sigma \star \eta$	`eta_tilde`	$\mathbb{F}_z^n$
Random masking vector	$\mathbf{u}'$	`u_tilde`	$\mathbb{F}_p^n$
Transformed masking vector	$\mathbf{u} = \mathbf{v} \star \mathbf{u}'$	`u`	$\mathbb{F}_p^n$
Syndrome (Public key)	$\mathbf{s} = \mathbf{e}\mathbf{H}^\top$	`s`	$\mathbb{F}_p^{n-k}$
Transformed syndrome	$\tilde{\mathbf{s}}$	`s_tilde`, `res`	$\mathbb{F}_p^{n-k}$
Reduced matrix from $\mathbf{H}^\top$	n/a	`v_tr`	$\mathbb{F}_p^{k\times(n-k)}$

Table 3.1 Mapping of Specification and Code Variables in CROSS

This mapping aids in navigating between the algebraic objects in the CROSS specification and their implementation counterparts in code. The code uses the suffix `_tilde` to refer to pre-transformation variables, which correspond to primed notation (e.g., e') in the specification.

The following parameters of CROSS are described in the specification and depend on the configuration options introduced above. This work focuses on *CROSS-R-SDP-1 fast* and thus uses the following parameter values:

- $p = 127$: element types for field $\mathbb{F}_p^n$
- $z = 7$: order of the cyclic subgroup $\mathbb{E} = \langle g \rangle \subset \mathbb{F}_p^*$
- $n = 127$: length of linear code and thus of vectors $\mathbf{u}$, $\mathbf{u}'$, $\mathbf{e}$ and $\mathbf{e}'$
- $k = 76$: dimension of the linear code, $k < n$
- $t = 163$: denotes the number of *rounds*
- $w = 85$: weight, i.e., number of rounds with second challenge $b = 1$

Note that $z = 7$ implies that the cyclic subgroup $\mathbb{E} = \langle g \rangle$ has order z and is embedded in $\mathbb{F}_p^*$, the multiplicative group of nonzero elements in $\mathbb{F}_p$. Elements of G lie in $\mathbb{E}^n$, and operations involving η, σ and $\mathbf{v}$ are performed modulo z.

3.2 Attack Fundamentals

3.2.1 Noise on Microcontrollers

In practice, real microcontroller units (MCUs) like the STM32F4 exhibit various noise sources that complicate leakage extraction. Overlapping components of power consumption can arise from:

- parallel operation of the CPU core, ALU, control logic, and bus arbitration;
- memory and cache accesses (SRAM, Flash);
- data bus transitions and register-to-memory communication;

These effects reduce the signal-to-noise ratio and make it more difficult to isolate leakage from a specific operation. Furthermore, clock-domain jitter (small, often random variations in the clock signal) can introduce timing misalignment across traces, degrading the effectiveness of averaging or profiling techniques.

3.2.2 Leakage Model

The STM32F4, like most modern embedded devices, is based on dynamic CMOS technology. The power consumption of such microcontrollers does not correlate directly with the numeric values being processed, but rather with their binary representation, i.e., the number of bits set to one. Accordingly, a Hamming weight leakage model is commonly used for side-channel analysis of MCUs, as it provides a reasonable approximation of the instantaneous power consumption during data processing. This model is particularly accurate when values are written to registers, since dynamic power consumption in CMOS is closely tied to the number of bit transitions from zero to one.

Throughout this work, we refer to the original integer values used in leakage modeling as *labels*, and to the corresponding Hamming weights of these values as *Hamming weight labels*. The latter are used to train statistical models during the profiling phase of the template attack.

This work focuses on first-order leakage, modeled through the Hamming weight of single intermediate values. Higher-order effects, as described in subsection 3.2.1, are not considered. Additionally, leakage due to memory accesses—for instance, when values are loaded into registers—may reveal further information beyond computed results.

3.2.3 Tools: SCALib, SNR metrics and LDA modeling

In this thesis, we use SCALib [2], a Python-based library offering tools that support side-channel analysis. It includes tutorials, such as a basic template attack example on AES, to help users get started.

The signal-to-noise ratio (SNR) metric quantifies how much information about a random variable is present in the mean of a set of trace samples. Higher SNR values indicate stronger correlations between the traces and the processed data, helping to identify sample points useful for side-channel analysis. We use SCALib's SNR computation to identify points of interest (POIs) by correlating trace samples with known values. These are derived under a Hamming weight leakage model and represent the Hamming weights of selected intermediate values. Each intermediate value is typically associated with multiple POIs. We refer to the collection of POIs linked to one intermediate as a *POI set*, where each POI is a single time sample, and the full POI set captures the relevant leakage window for that value.

To model the leakage behavior, we apply SCALib's linear discriminant analysis (LDA) tools. LDA is a statistical method that finds linear combinations of features that best separate data classes. In our case, those classes represent distinct Hamming weight values. The model is trained using the previously identified POIs and the Hamming weight labels of the known intermediates. The quality of the LDA model benefits from diverse and abundant training data. Once trained, the model can be applied to traces from the target device to estimate a probability distribution over possible Hamming weights of unknown values.

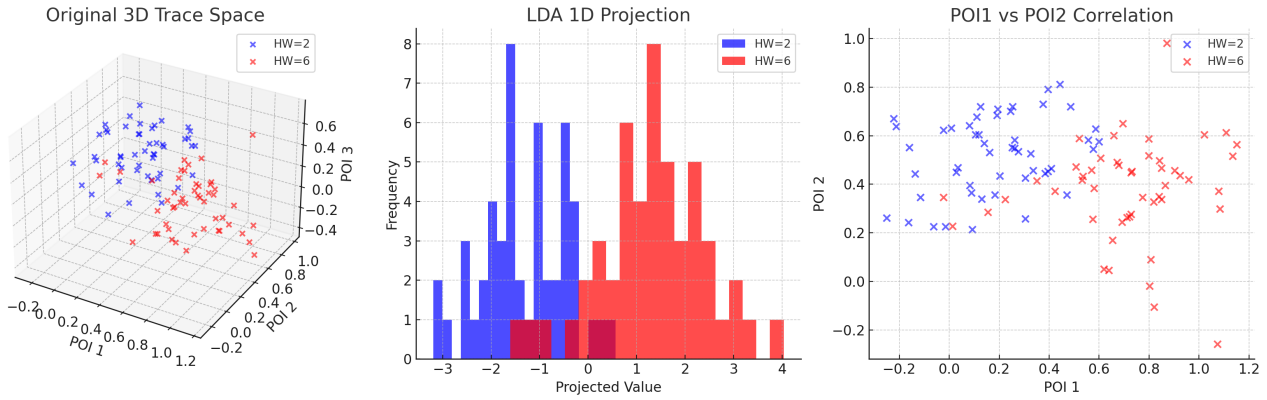


Figure 3.1 LDA projection 3D to 1D space

Figure 3.1 illustrates how LDA operates when using three POIs, which form a 3-dimensional input space. Each trace corresponds to a point in this space, and the labels represent Hamming weight classes—in this example, HW=2 and HW=6.

LDA computes the mean vector of each class and identifies a projection axis (a linear combination of POI1, POI2, and POI3) that maximizes the ratio of between-class variance to within-class variance. This projection direction represents the axis along which the classes are best separated. Each 3D trace sample is then projected onto this 1D axis, resulting in scalar values that cluster around class-specific means. These projected values serve as scores used to distinguish between Hamming weight classes in actual attacks.

Also shown in the figure is a correlation plot (e.g., POI1 vs. POI2), which visualizes how POI samples relate across traces. If two POIs are highly correlated across all traces (i.e., their values follow a similar linear trend), they may carry redundant information. In contrast, low inter-POI correlation implies that each POI captures different aspects of the leakage, potentially enhancing the discriminative power of the LDA model.

In this specific example, POI1 (horizontal axis) shows a clearer separation between HW=2 and HW=6 than POI2 (vertical axis), indicating that POI1 contributes more strongly to distinguishing these classes. The correlation plot shows that although HW=2 and HW=6 form distinct clusters, their alignment does not follow a strong diagonal trend through the origin, suggesting limited linear redundancy between POI1 and POI2.

3.2.4 Template Attacks

Template attacks are a powerful class of profiling-based side-channel attacks. Their core idea is to model the power consumption behavior of a device under controlled conditions and later use this model to infer secrets from an unknown device of the same type.

In our setting, a *k-trace attack* refers to an attack that recovers the full secret using k signature traces. In the best case, template attacks can achieve *1-trace attacks*, i.e., they succeed using a single signature trace (or equivalently, a group of partial traces linked to that signature). While this is achievable in many cases, real-world performance can depend heavily on the underlying intermediate values and noise conditions.

In the *profiling phase*, the attacker uses a device they fully control to execute cryptographic operations with known intermediate values, often derived from known inputs and keys (referred to as *labels*). For each execution, the power consumption is recorded as a trace. These traces are statistically analyzed to identify time samples that correlate strongly with the labeled values. These key time samples are referred to as POIs.

Based on the labeled traces and selected POIs, the attacker constructs a statistical model (often multivariate Gaussian) that captures the device's leakage characteristics with respect to the chosen intermediate values.

In the subsequent *attack phase*, the target device performs the same operations, but with an unknown key. The attacker records a small number of power traces and applies the previously constructed model to compute the likelihood of each possible value of the targeted secret. This enables efficient key recovery, especially when the leakage model and POI selection are accurate.

Template attacks require more control over the profiling device than DPA, but they can be extremely effective when assumptions about leakage and device similarity hold.

3.2.5 Divide-and-Conquer

SCAs often exploit the independence between different components of a secret, enabling a *divide-and-conquer* strategy. Instead of recovering the entire key in one step, the attack is decomposed into multiple smaller recoveries—for instance, per-byte, per-element, or per-round. This reduces the size of the hypothesis space and allows exhaustive hypothesis testing to be applied independently across segments.

3.2.6 Non-Linearity and Intermediate Operations

In side-channel attacks, non-linear operations often amplify the exploitable differences between secret-dependent computations. This principle underpins the effectiveness of classical attacks on AES, where the S-Box introduces significant non-linearity and dispersion in intermediate values.

This is particularly relevant when analyzing intermediate computations rather than secret values themselves. In many cryptographic implementations, secret-dependent variables are used as operands in arithmetic operations—such as integer multiplication or addition—before being reduced or mapped back into a smaller field. These intermediate results can reveal significantly more information than the original secret operands. Even if two candidate secrets share the same Hamming weight, their participation in a non-linear operation can produce distinct results with different leakage characteristics.

Thus, from an attacker's perspective, it is often more effective to construct hypotheses on such intermediate values. Doing so leverages the non-linearity of the operations to break ambiguities that would persist if the secret value were attacked in isolation. This insight is fundamental to many divide-and-conquer strategies in side-channel attacks, where the goal is to isolate and exploit high-leakage intermediates rather than directly infer low-leakage secrets.

Example 3.2.1 (Effect of Operations on Hypothesis Distinguishability) Assume two secret values:

$$s_1 = 0b00101101 \quad (\mathrm{HW}(s_1) = 4), \quad s_2 = 0b00011110 \quad (\mathrm{HW}(s_2) = 4)$$

Under the Hamming weight linear leakage model, these are indistinguishable. Now consider using each in an arithmetic operation with a known operand (e.g., multiplication with $a = 3$), and observing the Hamming weight of the result:

Secret	HW(secret)	Result of $3 \cdot s$	HW(result)
$s_1 = 45$	4	$3 \cdot 45 = 135$	HW(135) = 4
$s_2 = 30$	4	$3 \cdot 30 = 90$	HW(90) = 3

While s_1 and s_2 are indistinguishable by their Hamming weights alone, their use in a multiplication can result in outputs with different Hamming weights. This difference improves the distinguishability of hypotheses during profiling or classification.

3.2.7 Digital Signature Schemes

Digital signature schemes consist of three fundamental operations: *key generation*, *signing*, and *verification*. The first two are performed by the signer (i.e., the prover $\mathcal{P}$) and require access to the secret key, while verification is carried out by the verifier using only the public key.

- `KeyGen()` $\rightarrow$ (`pk`, `sk`) // Generate a public-secret key pair
- `Sign`(`sk`, `msg`) $\rightarrow$ `sig` // Compute a signature for the message
- `Verify`(`pk`, `msg`, `sig`) $\rightarrow$ {accept, reject} // Check the signature's validity

Key generation is typically performed only once per signer, whereas the signing operation is executed repeatedly and is therefore of particular interest for SCAs. Accordingly, this thesis focuses on analyzing and attacking the signing operation.

In the case of CROSS, the `KeyGen` function takes a uniformly random *Seed* as input. This *Seed* deterministically defines the secret key and is used to derive the public key via a cryptographically secure pseudorandom number generator (CSPRNG).

3.3 Analysis of the CROSS Algorithm

3.3.1 CROSS Specification: Vulnerability and Attack Surface Analysis

Algorithm 1, adapted from the official CROSS specification, and Algorithm 2 in the specification itself describe how the CROSS-ID protocol operates in *rounds* during signature generation.

The secret key consists of a vector $\mathbf{e}$, while the public key includes a parity-check matrix $\mathbf{H}$, a group G, and a syndrome $\mathbf{s}$. In the CROSS specification, the secret key is formally sampled from a group $\mathbb{G} \subseteq \mathbb{E} \subset \mathbb{F}_p^*$. In case of CROSS-R-SDP we have $\mathbb{G} = \mathbb{E}$, and thus the full subgroup is used for key sampling. Only in the CROSS-R-SDP(G) variant is $\mathbb{G}$ a proper subgroup of $\mathbb{E}$, used to reduce the key space and constrain decoding difficulty.

Algorithm 1 The CROSS-fast signature scheme: signature generation. Adapted from Figure 4 of the CROSS specification.

Private key: $\mathbf{e} \in G$
Public key: $G \subset \mathbb{E}^n, \mathbf{H} \in \mathbb{F}_p^{(n-k)\times n}, \mathbf{s} = \mathbf{e}\mathbf{H}^\top \in \mathbb{F}_p^{n-k}$
Input: Message $\mathtt{Msg}$
Output: Signature $\mathtt{Signature}$

Sample $\mathtt{Salt} \xleftarrow{\$} \{0,1\}^{2\lambda}$

For $i = 1, \ldots, t$:
  Sample $\mathtt{Seed}^{(i)} \xleftarrow{\$} \{0,1\}^{\lambda}$
  Sample $(\mathbf{e}'^{(i)}, \mathbf{u}'^{(i)}) \xleftarrow{\mathtt{Seed}^{(i)}} G \times \mathbb{F}_p^n$
  Compute $\sigma^{(i)}$ such that $\sigma^{(i)}(\mathbf{e}'^{(i)}) = \mathbf{e}$
  Set $\mathbf{u}^{(i)} = \sigma^{(i)}(\mathbf{u}'^{(i)})$
  Compute $\tilde{\mathbf{s}}^{(i)} = \mathbf{u}^{(i)}\mathbf{H}^\top$
  $\mathsf{c}_0^{(i)} = \mathsf{Hash}(\tilde{\mathbf{s}}^{(i)}, \sigma^{(i)}, \mathtt{Salt}, i)$
  $\mathsf{c}_1^{(i)} = \mathsf{Hash}(\mathbf{u}'^{(i)}, \mathbf{e}'^{(i)}, \mathtt{Salt}, i)$

Compute $\mathsf{c}_0 = \mathsf{Hash}(\mathsf{c}_0^{(1)}, \ldots, \mathsf{c}_0^{(t)})$
Compute $\mathsf{c}_1 = \mathsf{Hash}(\mathsf{c}_1^{(1)}, \ldots, \mathsf{c}_1^{(t)})$
Compute $\mathsf{c} = \mathsf{Hash}(\mathsf{c}_0, \mathsf{c}_1)$

Generate $\beta^{(1)}, \ldots, \beta^{(t)} = \mathsf{GenCh1}(\mathsf{c}, \mathtt{Msg}, \mathtt{Salt})$
For $i = 1, \ldots, t$:
  Compute $\mathbf{y}^{(i)} = \mathbf{u}'^{(i)} + \beta^{(i)} \cdot \mathbf{e}'^{(i)}$
  Compute $h^{(i)} = \mathsf{Hash}(\mathbf{y}^{(i)})$
Compute $h = \mathsf{Hash}(h^{(1)}, \ldots, h^{(t)})$

Generate $b^{(1)}, \ldots, b^{(t)} = \mathsf{GenCh2}(\mathsf{c}, \beta^{(1)}, \ldots, \beta^{(t)}, h, \mathtt{Msg}, \mathtt{Salt})$
For $i = 1, \ldots, t$:
  If $b^{(i)} = 0$:
    Set $f^{(i)} := (\mathbf{y}^{(i)}, \sigma^{(i)}, \mathsf{c}_1^{(i)})$
  Else:
    Set $f^{(i)} := (\mathtt{Seed}^{(i)}, \mathsf{c}_0^{(i)})$

Set $\mathtt{Signature} := \{\mathtt{Salt}, \mathsf{c}, h, f^{(1)}, \ldots, f^{(t)}\}$

During signing, the signer, i.e., the prover $\mathcal{P}$, samples a fresh *round seed* (denoted $\mathtt{Seed}$ in Algorithm 1) using a true random number generator (TRNG). This should not be confused with the fixed *Seed* used to derive the scheme's secret and public keys. The round seed is then used with a CSPRNG to generate the vectors $\mathbf{e}'$ and $\mathbf{u}'$.

A transformation vector σ is computed in exponent space such that $\eta = \sigma + \eta' \mod z$, where η and η' represent the exponent forms of $\mathbf{e}$ and $\mathbf{e}'$, respectively. The exponent vector σ is then converted to a multiplicative representation $\mathbf{v} \in \mathbb{F}_p^n$, which is used to compute $\mathbf{u} = \mathbf{v} \star \mathbf{u}'$ and $\mathbf{e} = \mathbf{v} \star \mathbf{e}'$.

A key observation is that each element of the vector $\mathbf{u} \in \mathbb{F}_p^n$ influences $n - k$ output elements via the vector-matrix product $\mathbf{u}\mathbf{H}^\top$, effectively being reused across multiple computations with known values from $\mathbf{H}^\top$.

Although $\mathbf{u}$ is not directly tied to the secret key $\mathbf{e}$, it is linked to intermediate values that expose information about $\mathbf{e}$. In rounds where the challenge bit is $b = 1$, the signature reveals the round seed, allowing us to reconstruct both $\mathbf{e}'$ and $\mathbf{u}'$. If $\mathbf{u}$ can be recovered via side-channel analysis, then σ can be derived from $\mathbf{u}'$, and subsequently σ can be used to compute $\mathbf{e}$ from $\mathbf{e}'$—thereby recovering the secret key. In contrast, for the less frequent rounds where $b = 0$, the signature includes $\mathbf{y}$ and σ instead of

the round seed. While key recovery is still theoretically possible via these values, it requires a different attack strategy and is complicated by their lower frequency in many CROSS variants.

None of the other critical values are processed as often as $\mathbf{u}$ in the multiplication with $\mathbf{H}^\top$. In particular, $\mathbf{e}$, $\mathbf{e}'$, σ, $\mathbf{u}'$ are each used only once per round. The repeated processing of $\mathbf{u}$ provides more opportunities for leakage and thus for constructing effective hypotheses.

We can use hypotheses on single elements of $\mathbf{u} \in \mathbb{F}_p^n$ because they are processed independently of each other, which causes the hypothesis space for each separate value in $\mathbf{u}$ to be p. By posing hypotheses separately for n single elements of $\mathbf{u}$, we can we can derive hypotheses for the corresponding intermediate values produced by the vector-matrix multiplication. This follows the divide-and-conquer approach and limits the hypothesis space of each element to p values. For all configurations of CROSS these values are within a range that can be tested computationally.

The ultimate goal of our attack is to recover the long-term secret vector $\mathbf{e}$. While the long-term *Seed* (used in key generation) is never exposed directly, the vectors $\mathbf{e}$ and its exponent form η are derived from it and remain fixed for a given signer. Recovering these derived values is sufficient to generate valid signatures.

To illustrate the recovery path, we consider the following transformations:

The ultimate goal of our attack is to recover the long-term secret vector η...

$$
\begin{aligned}
\text{(forward)} \quad & \sigma = \eta - \eta' \mod z \\
& \mathbf{v} = \text{convert}(\sigma) \in \mathbb{F}_p^n \\
& \mathbf{u} = \mathbf{v} \star \mathbf{u}' \mod p \\
\text{(backward)} \quad & \mathbf{v} = \mathbf{u} \star {\mathbf{u}'}^{-1} \mod p \\
& \sigma \Leftarrow \mathbf{v} \\
& \eta = \sigma + \eta' \mod z
\end{aligned}
$$

Figure 3.2 Forward and backward transformations for recovering the long-term secret.

The recovery of $\mathbf{u}$ from $\mathbf{u}'$ and η is always possible. On the other hand, recovering η from $\mathbf{u}$ and $\mathbf{u}'$ is only possible if the intermediate value $\mathbf{v}_i = \mathbf{u}_i \cdot (\mathbf{u}'_i)^{-1} \mod p$ lies in the multiplicative subgroup $\mathbb{E} \subset \mathbb{F}_p^\times$ corresponding to the isometry. This requires $\mathbf{v}_i$ to be a power of 2. In practice, this condition fails if either $\mathbf{u}_i = 0$ or $\mathbf{u}'_i = 0$, which occurs in approximately $2/p$ of the cases for uniformly random vectors in $\mathbb{F}_p^n$.

3.3.2 Source Code Analysis

To better understand the real behavior of the signing implementation, we now analyze the reference C source code and highlight how it differs from the high-level specification to improve computational efficiency and reduce memory usage.

Code Structure and Correspondence to Specification. Specifically, the implementation uses a reduced form of the transposed matrix $\mathbf{H}^\top$, denoted `V_tr` $\in \mathbb{F}_p^{k\times(n-k)}$, rather than the full transpose $\mathbf{H}^\top \in \mathbb{F}_p^{(n-k)\times n}$, thereby omitting the identity submatrix. Accordingly, the vector-matrix multiplication is not performed explicitly using all columns of $\mathbf{H}^\top$; instead, a more efficient form is used. Listing 3.1 shows a slightly modified version of the vector-matrix multiplication implementation in the original source code.

The result vector, denoted `s_tilde` in the specification and `res` within the source code function for the multiplication routine, is initialized such that its first $n-k$ entries are set to the last $n-k$ values of the input vector `u`. The outer loop iterates over indices i of the vector `u`, while the inner loop steps through j to access entries in `V_tr[i][j]`. Each inner loop execution performs a multiplication, an addition and a reduction.

Listing 3.2 displays the assumed order of operations with intermediate results as we would expect them to be calculated on a 32-bit CPU. This decomposition helps isolate points of interest for power analysis, especially the multiplication and addition steps.

After priming `res[0:N-K-1]` with `u[K:N-1]`, each of the first k elements of `u` is multiplied by $n-k$ entries of the corresponding row in `V_tr`. The results of these individual multiplications are referred to as *intermediate 1* or `im1` values. These values are then added to entries from the `res` vector, producing a new set of values called *intermediate 2* or `im2`. Each `im2` value is then reduced modulo p (i.e., in $\mathbb{F}_p$), and `res` is updated accordingly.

```
#define FQRED_SINGLE(x) (((x) & 0x7F) + ((x) >> 7))
#define FQRED_DOUBLE(x) FQRED_SINGLE(FQRED_SINGLE(x))

void fq_vec_by_fq_matrix(uint8_t res[N-K], uint8_t u[N], uint8_t V_tr[K][N-K]) {
  memcpy(res,u+K,(N-K)*sizeof(uint8_t));
  for(int i = 0; i < K; i++){
    for(int j = 0; j < N-K; j++){
      // multiply & accumulate
      uint16_t tmp = (uint16_t)res[j] + (uint16_t)u[i] * V_tr[i][j];
      // fast + lazy modulo-127 reduction via bitmasking and shifts
      res[j] = FQRED_DOUBLE(tmp);
    }
  }
}
```

Listing 3.1 Vector-matrix multiplication for CROSS-R-SDP, with explicit types and reduction step separated

```
void fq_vec_by_fq_matrix(uint8_t res[N-K], uint8_t u[N], uint8_t V_tr[K][N-K]) {
  memcpy(res,u+K,(N-K)*sizeof(uint8_t));
  for(int i = 0; i < K; i++){
    for(int j = 0; j < N-K; j++){
      // set trigger signal
      uint16_t im1 = (uint16_t)u[i] * V_tr[i][j]; // POI candidate
      uint16_t im2 = (uint16_t)res[j] + im1; // POI candidate
      uint8_t im3 = im2 & 0x7F;
      uint8_t im4 = im2 >> 7;
      uint8_t im5 = im3 + im4;
      uint8_t im6 = im5 & 0x7F;
      uint8_t im7 = im5 >> 7;
      res[j] = im6 + im7;
      // reset trigger signal
    }
  }
}
```

Listing 3.2 Vector-matrix multiplication for CROSS-R-SDP, with assumed intermediate results and trigger

Detailed analysis. To attack the vector `u[0:N-1]`, we look at operations where elements of `u` are processed. We can see two operations where this is the case: the first is the multiplication resulting in `im1`; the second is the addition leading to `im2`, which involves `im1` and an element of res[0:N-K-1].

Thus, the first *batch* of `u`, i.e., `u[0:K-1]`, has leakage in `im1`. And *batch 2*, i.e., `u[K:N-1]`, affects `im2`. `im1` and `im2` are the processed values which we target as hypotheses derived from values of `u`. To evaluate those intermediate values, we need to consider the following properties of the matrix multiplication:

- sets of $n-k$ `im1` values depend only on a single unknown value of `u[0:K-1]` and known values from `V_tr`

- each `im2` does not only depend, through `res`, on a value from the second batch of `u`, but via `im1` also on one `u[0:K-1]`
- `res[j]` are initialized with values from `u`, but are continuously updated and reused; this needs to be considered to follow derived hypotheses
- each `im2` reflects contributions from one value from `u[K:N-1]` (via `res`) and one value from `u[0:K-1]`
- thus, a set of k `im2`-values which depends on the same element of `u[K:N-1]` additionally depends on k values of `u[0:K-1]`; i.e., to use those k `im2` together for targeting one value of `u[K:N-1]`, the k values of `u[0:K-1]` need to be considered

As a result, to turn a hypothesis for one element of `u[K:N-1]` into k hypotheses for correspondong `im2` values comprises k values of the so far unknown first segment of `u[0:K-1]`. This increases the hypothesis space to an infeasible number of values for the second `u`-batch.

Adapted attack strategy. To overcome this problem, we deviate from the strategy outlined in subsection 3.3.1 by dividing the attack into two batches based on the dependency structure of the vector-matrix multiplication. We recover the values in `u[0:K-1]` first by targeting `im1`. These values can then be treated as known in a second stage of the attack, thus eliminating the need to hypothesize the first k values of `u` and bringing the size of the hypothesis space for each element of u back to p.

For the first k elements of `u`, the attack proceeds as previously described. Each value in `u[0:K-1]` is used in $n-k$ multiplications with known constants from the public matrix `V_tr`, resulting in $k(n-k)$ total `im1` values. This repetition enables a divide-and-conquer approach: each `im1` value depends on only one unknown from `u` and a known constant, making it feasible to test all hypotheses over $\mathbb{F}_p$.

For the second batch of `u`, i.e., `u[K:N-1]` the attack becomes more complicated, because its values are used in the initialization of `res`, which is updated in-place and involved in subsequent `im2` computations. In total, there are again $k(n-k)$ additions resulting in `im2` values. Assuming the `im1` values are known, the remaining unknowns are the $n-k$ entries in `u[K:N-1]` and the corresponding dynamic values in `res`. For each outer iteration index i and inner index j, the attack must account for the dependencies introduced by the update structure of `res[j]`, which influences the computation of future `im2` values. These data dependencies must be carefully modeled when constructing hypotheses over the remaining part of `u`.

This staged recovery preserves the tractability of the attack.

Lazy Modular Reduction. In all CROSS-R-SDP variants, the modular value for the Galois field is $p = 127$. This is a Mersenne number, i.e., a number of the form $2^n - 1$, and also a Mersenne prime. The C implementation for CROSS-R-SDP variants uses a form of lazy modular reduction which is shown in Listing 3.1. This reduction works for such Mersenne numbers, but does not reduce all values to $GF(127)$—the value $p = 127$ can occur after the lazy reduction takes place. In the field $\mathbb{F}_{127}$, we observe that $127 \equiv 0 \pmod{127}$. This implies:

$$x + 127 \equiv x + 0 \equiv x \pmod{127}, \quad \text{and} \quad x \cdot 127 \equiv x \cdot 0 \equiv 0 \pmod{127}$$

Hence, even if intermediate values like $\texttt{res}[j] = 127$ appear in the implementation due to lazy reduction, they are mathematically valid representatives of the field element 0. This congruence implies that arithmetic correctness is preserved as long as all subsequent operations are interpreted modulo 127.

In practice, this lazy reduction is exploited for efficiency. The system may store or propagate the unreduced value 127 (instead of explicitly reducing it to 0), since further field operations naturally bring values back into the proper domain.

Unless explicitly stated otherwise, the term *reduction* in this thesis refers to the *lazy reduction* approach used in the implementation.

3.4 Practical Constraints

3.4.1 Target Device Limitations

We need to consider the limitations of the DUT and the memory footprint of the reference implementation. Without dedicated source code optimizations, the STM32F4 target is unable to execute the full signing operation of CROSS-*R-SDP-1-fast* due to memory constraints. Tests with modified source code showed that the target can reliably execute a single round of the CROSS-ID protocol (referred to as *round 0*). We do not know the exact maximum number of rounds it could handle, but a single round is sufficient for our needs. Instead of working with complete signing operations, we collect attack traces from repeated executions of round 0. Each of these corresponds to one vector-matrix multiplication and results in $k(n-k)$ scalar operations.

The complete signing operation (used for signature generation or verification) can still be executed off-device using an x86 build of CROSS. Even though the current CROSS reference implementation uses only a CSPRNG, a practical implementation would later use a TRNG for sampling. This does not obstruct our approach: for profiling, we can still use a CSPRNG; and while attack traces from a realistic device will be obtained from a CROSS implementation using a TRNG, this does not change the feasibility of the attack method.

We use the same inputs for profiling and attack traces (i.e., $Seed$ and Msg), enabling deterministic execution on the DUT through a CSPRNG. This ensures the exact same values for `u`, `u_tilde`, and `e_tilde`. When applied to a full, instrumented implementation of CROSS, these inputs also yield a valid signature and allow extraction of `eta_tilde` and `u_tilde` (for $b = 1$ rounds), along with internal values `im1` and `im2`.

This motivates our trace collection strategy: a regular signature for CROSS-*R-SDP-1-fast* yields $t = 163$ rounds, of which $w = 85$ use $b = 1$ and can be exploited in the attack. To obtain an equivalent dataset without full signing on the device, we repeatedly invoke partial signing with only round 0 and collect 85 first rounds where $b = 1$. This yields the same number of usable attack rounds as a full signature-based trace.

Unless explicitly stated otherwise, the term *traces* in this work refers not to full signature traces with $t = 163$ rounds (as used in a complete signing operation of CROSS-R-SDP-1-fast), but to single-round traces obtained from isolated executions of round 0.

3.4.2 Variance in Traces

The CROSS v1.2 reference implementation relies on a deterministically seeded CSPRNG for all randomness to allow deterministic executions for testing. As a result, simply varying the message or the *Seed* value does not introduce sufficient variability into the sampled values during signature generation.

To ensure variance across profiling and attack traces, we use a different message for each trace. Using the message to seed the CSPRNG guarantees that the randomly sampled values $\mathbf{u}'$ and $\mathbf{e}'$ differ across traces, while maintaining deterministic and reproducible behavior for each trace.

This approach simulates the variance that would naturally occur in a real-world implementation, where a TRNG would be used to sample fresh randomness on each signature attempt. It also enables precise trace alignment and analysis by preserving reproducibility of intermediate values across repeated executions.

3.5 Attack Optimizations and Design Decisions

3.5.1 Reduction of Hypothesis Space

In our attack scenario, we assume that a valid signature is available, allowing us to reconstruct the values of $\mathbf{u}'$ and η'. While the isometry vector $\sigma \in \mathbb{F}_z^n$ is unknown, we know that each of its components can only take one of z distinct values, where $z \ll p$.

Since $\mathbf{u} = \mathbf{v} \star \mathbf{u}'$ with $\mathbf{v}$ derived from σ, we can generate z candidate values for each element $\mathbf{u}_i$ by iterating over all possible σ_i and computing $\mathbf{u}_i = \mathbf{v}_i \cdot \mathbf{u}'_i$.

This reduces the hypothesis space for each $\mathbf{u}_i$ from p to z values. As a result, the number of candidate values to test is significantly smaller, reducing computational effort and allowing early elimination of incorrect hypotheses.

This improvement is limited by the fact that the possible values of $\mathbf{v}$ are powers of 2. Multiplying any fixed value $\mathbf{u}'_i$ with a power of 2 corresponds to a left-shift in binary representation, which generally preserves the Hamming weight of the result. As a consequence, all z hypotheses for a given $\mathbf{u}_i$ may belong to the same Hamming weight or LDA class, making them indistinguishable in the profiling model.

A distinction is only possible when the product $\mathbf{v}_i \cdot \mathbf{u}'_i$ exceeds $p - 1$, triggering modular reduction in $\mathbb{F}_p$. In this case, the reduction can change the Hamming weight of the result, allowing at least some hypotheses to fall into a different Hamming weight class.

The reduction function used in the reference implementation mimics modulo-127 via repeated bit-masking and addition: as long as the product is below 127, no reduction occurs and the Hamming weight remains unchanged. Thus, hypothesis distinguishability depends on whether reduction is triggered by the specific combination of $\mathbf{v}_i$ and $\mathbf{u}'_i$.

3.5.2 Distribution of Hamming Weights and Handling of Rare LDA Classes

When analyzing intermediate values such as `im1` and `im2`, we observed that the distribution of resulting Hamming weights (HW) is highly imbalanced. This poses a challenge for profiling-based attacks like LDA, which require sufficient training data for each class.

We first empirically verified the distribution of Hamming weights for `im1` and `im2` values by analyzing all combinations of values occurring in multiplication and addition steps without modular reduction, as we expect them to occur during signature generation. Figure 3.3 compares the percentage of each Hamming weight derived from this simulation against the actual observed distribution in 20,000 measured traces.

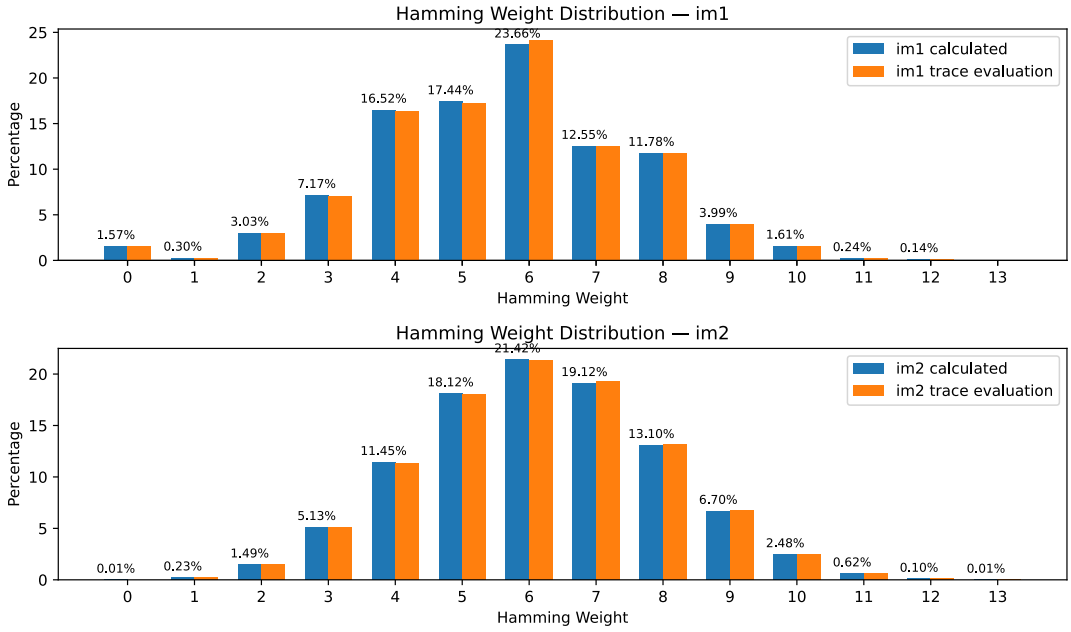


Figure 3.3 Comparison of Hamming weight distributions from calculated combinations and 20,000 trace file for `im1` and `im2`.

For `im1`, 13 Hamming weight classes occur (0 to 12), and for `im2`, 14 classes (0 to 13). However, the distribution is skewed: some classes such as HW=6 or HW=7 are common, while others like HW=0 or HW=13 are extremely rare or even absent from real traces.

SCALib's LDA models require each class label to be represented in the training data. To enrich underrepresented classes, we used a trial-and-error approach during trace acquisition. This resulted in a set of 5,514 *crafted traces* with specific inputs (Msg, $Seed$), specifically collected to ensure coverage of rare HW values in all intermediate columns. Those traces covered all HW classes for both `im1` and `im2`, except for HW=0 in `im2` which was too rare to appear reliably.

To address this, we effectively merged HW=0 with HW=1 for LDA purposes by shifting the class labels and remapping SCALib's class index 0 accordingly. This workaround ensures that all class indices required by the LDA model are populated, enabling stable training even without vertical stacking.

3.5.3 Vertical Stacking

Within one round, i.e., one vector-matrix multiplication, we observe $k(n-k)$ executions of the inner loop. Each loop iteration performs one multiplication, one addition, and a reduction.

The POIs for `im1` are expected to align with the trace samples corresponding to the multiplications, and those for `im2` with the additions. Each of the $k(n-k)$ `im1` samples and their corresponding labels can be considered results of the same operation. From the perspective of an LDA model, the order or position of these label-sample pairs is irrelevant.

After identifying the POIs via SNR analysis, we can align them by vertically stacking all POI samples for `im1` (and separately for `im2`) underneath each other. This enables training a single LDA model per label type, rather than $k(n-k)$ separate models. Each model thus receives $k(n-k)$ times more training data compared to training on only one POI set per label. This strategy mitigates the data imbalance problem, especially for rarely occurring Hamming weight classes. In addition, using fewer LDA models lowers memory requirements.

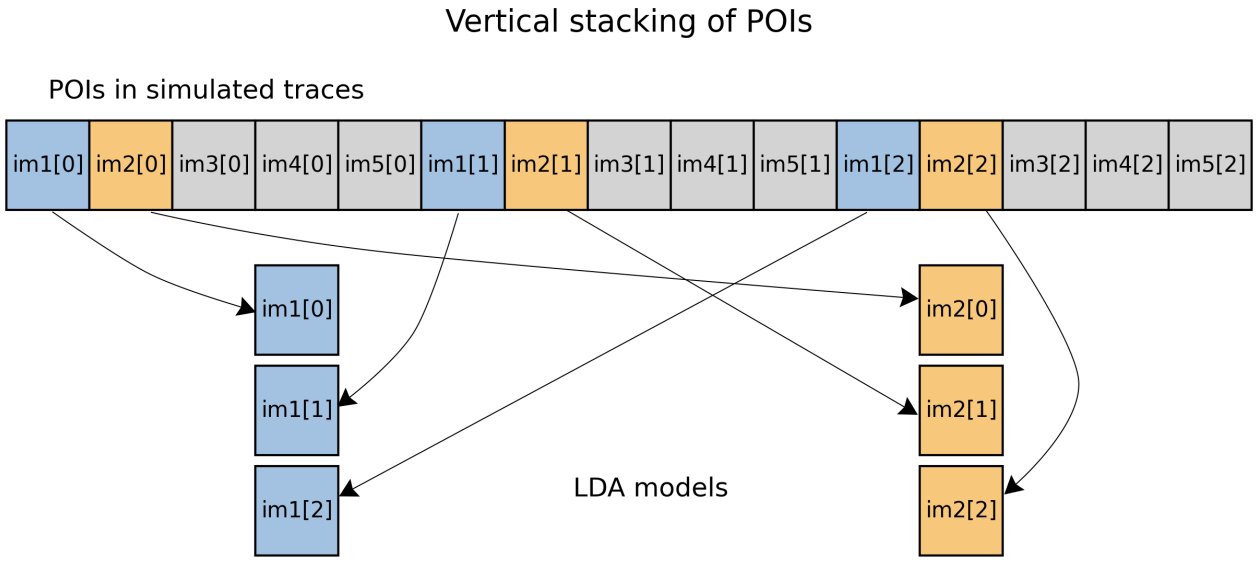


Figure 3.4 Vertical stacking of simulated trace POIs in LDA

3.5.4 Sample Batching for Large Trace Sets

Profiling traces can require substantial memory, especially when the number of samples per trace is high. For example, a single file with 20,000 traces may occupy approximately 800 GB of memory (for technical details, see subsection 3.7.1). This exceeds the available RAM even on high-performance machines, and thus prohibits direct full-trace processing.

To address this constraint, we split the processing into *sample batches*. This batching can be realized in two dimensions:

- **Vertical batching** keeps all traces but limits the number of samples per trace, i.e., only selected columns are loaded.
- **Horizontal batching** processes only a subset of traces (rows), but keeps all samples (columns).

Both strategies are supported in SCALib, since its SNR and LDA computations support incremental updates when data is reshaped into compatible formats. However, we follow a *vertical batching* strategy: at each step, a subset of samples from of traces is loaded and processed. We refer to such subsets as *sample batches*.

This choice is motivated by the structure of the SNR function, which computes column-wise statistics across traces and expects temporal alignment of data. If we want to locate POIs for a specific intermediate value—such as `im1` or `im2`—we expect this leakage to occur roughly at the same column offset across all traces. Horizontal batching would interfere with this alignment and potentially degrade POI detection accuracy.

Moreover, vertical batching is easier to integrate with both approaches: using *vertical-stacking* or using one LDA model per intermediate value. Horizontal batching would also be compatible with vertical stacking in principle, but it is harder to implement efficiently and would likely incur higher processing overhead.

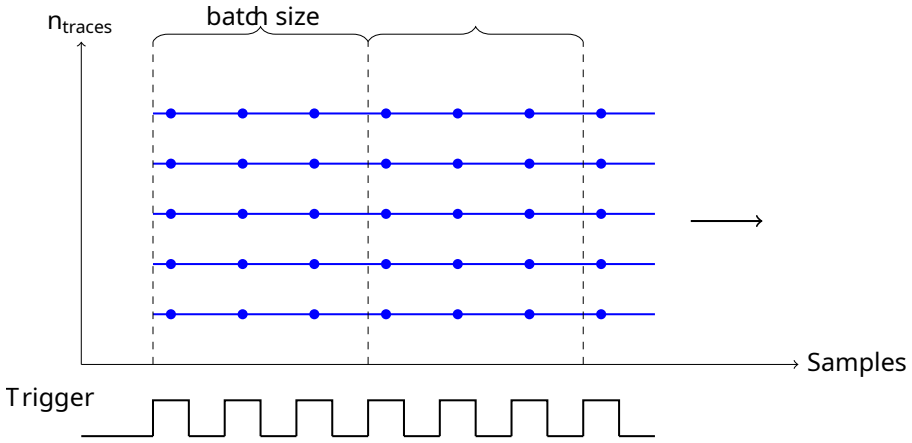


Figure 3.5 Sample batches with three POI sets per batch. Each column of blue dots, where a POI set is expected, lies within a trigger-identified window. Dashed lines denote vertical split boundaries.

Figure 3.5 illustrates a simplified example with two batches and three POI sets per batch. Here, the blue dots represent expected intermediate operations, aligned via trigger signals.

The batch size is defined by the number of intermediate values (POI sets) covered per pass. Trigger signals, recorded alongside the power trace, are used to isolate these regions by identifying high trigger intervals. Each high segment corresponds to an execution of the inner loop, and thus to one intermediate of interest.

Because the execution structure of CROSS is known (see Listing 3.2), we can reliably map specific trigger intervals to the computation of specific intermediates. This allows us to partition the trace into semantically meaningful windows and to limit POI search to those segments. This targeted approach significantly reduces computational overhead, as it avoids correlating labels with unrelated trace regions.

To ensure reliability, we experimentally verified that all trigger signals across our trace set were well-aligned and consistent in timing. This was critical to allow correct segmentation and ensure that POI searches across batches remained meaningful.

3.5.5 Reducing Storage Requirements

Template attacks often require a large number of profiling traces, which can result in significant storage demands. While full traces were saved to disk for post-processing and analysis, storage capacity and

network performance (e.g., when traces are stored on a NAS with limited throughput) introduced delays in trace acquisition and testing.

This constraint influenced both how traces were recorded and how intermediate results were reused or cached, aiming to reduce redundant access and limit I/O bottlenecks.

To reduce the storage footprint, a single digital trigger signal is stored in a separate file, rather than interleaving one trigger signal with each power trace. This effectively halves the trace file size. subsection 3.7.3 describes this in more detail.

3.6 Generation of Simulation Traces

To verify and test our attack strategies under controlled conditions, we generate simulated traces that emulate side-channel leakage from the vector-matrix multiplication in the signing algorithm. These traces help us evaluate the theoretical feasibility of our approach and understand the effect of leakage structure, class distribution, and preprocessing techniques (e.g., vertical stacking).

We denote with $r := n-k$ the number of intermediate values from the inner loop (`im1`, corresponding to the second batch of $\vec{u}$ values), and with $\ell := k(n-k)$ the total number of vector-matrix multiplications performed in the signing algorithm.

3.6.1 Trace Construction and Intermediate Extraction

We use the modified C implementation of the vector-matrix multiplication (see Listing 3.2) to compute realistic intermediate values (including `im1` and `im2`) for simulated traces.

In each iteration of the inner loop of the vector-matrix multiplication, five intermediate values are computed, including `im1` and `im2`. Since the loop runs ℓ times, this results in a total of $5 \cdot \ell$ intermediate values, i.e., simulated samples, per full round.

For profiling traces, both the secret key and the message input are varied for each trace. This allows the simulated leakage to span the full distribution of intermediate values, ensuring class coverage for training. For attack traces, however, we keep the secret key fixed and only vary the message input across traces. This mirrors the real-world scenario where the attacker targets a fixed key, and allows evaluation of recovery performance under realistic constraints.

Due to limitations of the DUT, only a single round can be executed for real trace collection. To maintain consistency with the profiling setup, simulated traces also use just one round of the matrix-vector multiplication per trace.

3.6.2 Trace Construction and Noise Modeling

To simulate side-channel leakage, we add Gaussian noise to the intermediate values, which represent idealized Hamming weights. The amount of noise is controlled by the standard deviation (std) of the Gaussian distribution. In this context, the std parameter reflects the noise level of the hypothetical measurement setup: smaller values model low-noise (e.g., close-proximity EM) measurements, while larger values correspond to higher-noise scenarios.

Each simulated trace includes $\ell = 3876$ intermediate computations, corresponding to the $k(n-k)$ scalar operations. For each vector-matrix multiplication, we create a fixed-length window of five samples, placed contiguously in time to model their layout in real traces. This results in $\ell \cdot 5 = 19380$ samples for each generated trace.

Each simulated intermediate value is first mapped to its Hamming weight, which serves as the leakage model. To introduce realistic variability, Gaussian noise is added to the Hamming weight before scaling and quantization. This simulates imperfect leakage and reflects measurement noise as observed in real power traces.

Reasonable levels for Gaussian noise can be derived from experience of researchers. In [9], the authors use a 32-bit STM32F405 microprocessor and measure noise with a standard deviation between 0.4 and 3.0.

Noise is, however, not the only factor distinguishing simulated from real-world traces. On actual microcontrollers, traces exhibit additional structure and variability due to device-specific effects, such as clock jitter, instruction overlap, and physical layout of data paths. These effects often introduce recurring patterns, misalignments, or localized leakage that are not captured by idealized simulation models. While Gaussian noise can approximate global randomness, it does not reflect these structured deviations, which can impact the detectability and stability of POIs.

3.7 Trace Capturing and Analysis

3.7.1 Measurement Setup

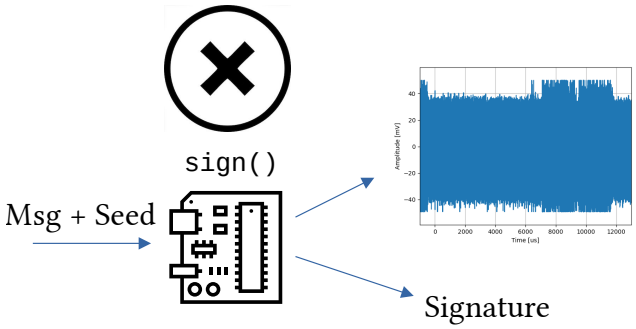


Figure 3.6 Measurement setup (partially adapted from SVG Repo, `https://www.svgrepo.com/svg/198478/circuit-board-microchip`, CC0 license).

To collect power traces, we use a shunt resistor setup in combination with a PicoScope DSO. The oscilloscope is controlled by a Python script, which writes obtained traces to Hierarchical Data Format 5 (HDF5) files for downstream processing.

The PicoScope supports a maximum sampling rate of 2.5×10^9 samples/s, while the target STM32F4 microcontroller operates at 10 MHz. For the default CROSS compilation using size optimization (via the `-Os` compiler flag), this results in 250 samples per target clock cycle.

Each trace captures 16 ms, resulting in exactly 40×10^6 samples per trace. The sampling duration of 16 ms was determined empirically to cover the entire execution of round 0 in the target implementation.

Due to the use of high-pass filtering (DC block), the measured signal only reflects dynamic fluctuations in power, not the absolute consumption level. This allows isolation of switching activity, which correlates with Hamming weight transitions, even when the absolute change in value is small. The DSO captures traces with 8-bit sample resolution and is configured with a 50 mV amplitude limit.

3.7.2 Coordinated Trace Labeling via Target and Host Execution

Due to DUT contraints as described in subsection 3.4.1, like simulated traces, measured traces comprise only *round 0* of the CROSS-ID protocol. A measurement script uses the previously obtained inputs *Seed* and *Msg* (the crafted values described in subsection 3.5.2) and sends it to the DUT for execution. We modified the CROSS implementation for the target so that it (a) executes only the first round of the CROSS-ID protocol, (b) seeds the CSPRNG with the message, and (c) returns the values for `u` and `s_tilde` to the controlling Python script. This measurement script saves these values along with the inputs for the *signing* function (message and *Seed*) for each trace in the trace file.

To overcome the limitations of the DUT, we use a library build of the same CROSS configuration (and with the same modifications) on a host system. We call it, using the stored *Seed* and message, to execute complete signing operations and receive full signatures that correspond to inputs used on the target. This setup enables offline verification of the expected `im1` and `im2` labels by recomputing intermediate values using the stored inputs, following the known conventions for how *Seed* and the message influence randomness. Those signatures can be used as inputs for the *verification* function.

This allows us to retrieve corresponding values for `u_tilde` and `eta_tilde` for rounds where those are available ($b = 1$).

This coordinated use of the DUT and host environment allows us to collect real power traces from the target, while computing accurate intermediate labels offline using the same *Seed* and message. This ensures alignment between side-channel measurements and internal values such as `im1`, `im2`, `u_tilde`, and `eta_tilde`, even though they are not directly observable during DUT execution.

3.7.3 Trigger Analysis and Intervals

The profiling traces are collected using an additional digital trigger signal to mark the execution of the inner loop. This trigger is set high at the start and reset at the end of each iteration, as shown in Listing 3.2.

We use scripted analysis to verify alignment: (a) we check the distances and durations of trigger intervals within each trace, and (b) we compare trigger intervals across traces within the same file and across different trace files with the same configuration.

To ensure correct alignment, we verified that the trigger signal matched across traces by inspecting several randomly selected examples from different trace files and confirming consistency. The consistent alignment allows us to save and reuse a single reference trigger signal in a separate file from the power traces. This reduces storage demands by roughly a factor of two and simplifies signal processing, as described in subsection 3.5.5.

Although the trigger is digital in logic, it is physically an analog signal and does not switch instantaneously between low and high. To account for analog swing and oscillation at the transitions, we define and identify start and end of trigger toggles by using a minimum number of consecutive samples being high or low, respectively. To have a consistent alignment across and within traces, we align the identified trigger intervals to the nearest 250-sample boundaries, matching the DUT's 10 MHz clock cycle.

3.7.4 Preprocessing

As described above, our trace files contain measured values with 8-bit vertical resolution, stored as `int16` values. We do not apply amplitude normalization to the raw traces, as the relative amplitudes are consistent across measurements. However, we perform downsampling by averaging over a chosen number of samples, thereby creating new averaged samples. We use the term *dsf* as abbreviation for the *downsampling factor*, i.e., the number of samples to be averaged.

This reduces the trace size (in memory) and improve processing efficiency. The choice of the downsampling factor also has a noticeable impact on the quality of SNR results and can improve those results as compared to raw traces, e.g., by suppressing fluctuations around an amplitude value that provides higher correlations to the actual Hamming weight label than the fluctuating amplitudes.

Selecting an optimal downsampling factor for side-channel analysis often requires empirical testing. We limited the downsampling factors to be divisors of 250, which is the number of samples per target clock cycle.

When selecting POIs based on SNR, the number of POIs per intermediate value must be chosen with respect to the downsampling factor. If the number of POIs is too large relative to the length of the downsampled region, some POIs may fall outside the relevant interval. This can result in degraded model quality or misalignment of labels and samples.

3.7.5 Validating Data-Dependent Leakage via SNR

To validate the presence of data-dependent leakage, we compute the SNR at each sample point (after downsampling), using SCALib's SNR module. This analysis is performed separately for the `im1` and `im2` labels, using the aligned trigger intervals described in subsection 3.7.3.

SNR values vary across measurement sets and intermediate types. For the configuration used in this work (CROSS-R-SDP-1-fast), typical peak values are discussed in subsection 4.3.3. The peak values are distinctive from SNR values of non-matching trace samples, and allow for reliable selection of POIs.

3.8 Attack Procedure

Within the following section, we describe the full side-channel attack flow, from trace prediction and candidate scoring to hypothesis testing and final key reconstruction.

3.8.1 Prediction and Candidate Selection

Applying the LDA model to the attack traces and selected POIs, we obtain probability predictions for the Hamming weight of hypothesized `im1` and `im2` values. These intermediate values are derived from candidate values of $\mathbf{u}$, allowing us to assign probabilities to each candidate hypothesis for the elements of $\mathbf{u}$ and its corresponding η hypothesis; we compute the value of η_i corresponding to each $\mathbf{u}_i$ hypothesis via $\mathbf{u}'$ and η'.

While $\mathbf{u}$ differs between rounds, η is constant for a given signer, so we can leverage that: The probabilities of η hypotheses across rounds are added up to identify the most likely candidates using the hypothesis with the maximum likelihood sum.

3.8.2 Attack Parameter Configuration

This section introduces the key parameters used by the side-channel attack script.

The *downsampling factor* is described in subsection 3.7.4. It controls trace resolution and influences the time and memory needed to process traces. It can influence the SNR by reducing fluctuations. If the value is chosen too high, then information is lost.

`npoi` is the number of POIs, i.e., the number of samples, that are passed to the LDA model per *intermediate* value. This parameter needs to be chosen together with a sensible value for `dsf`: for high values of `dsf`, `npoi` should be tendentially smaller, and vice versa.

The POI strategy (`poi_strategy`) determines how POIs are selected: `centered` locates the POI peak and uses adjacent samples. `fixed` uses the absolute sample positions as found via SNR; those may not be contiguous.

`vstack`, introduced in subsection 3.5.3, determines whether the attack applies the vertical stacking strategy to exploit the available traces more efficiently and may help with reducing the necessary amount of profiling traces for a successful attack.

The parameter `LDA_p` controls the dimensionality of the projection space into which the original trace samples are reduced in LDA. Each intermediate value `im1` and `im2` is represented by `npoi` trace samples. The LDA classifier computes a projection from this `npoi`-dimensional space into a smaller space of `LDA_p` dimensions.

We use `ntraces_p` and `ntraces_a` to limit the number of actual used profiling and attack traces, respectively. They influence both the quality of the learned LDA model and the reliability of the attack success rate evaluation.

We use `ntraces_p` and `ntraces_a` to limit the number of profiling and attack traces used, respectively. This determines how many traces are loaded and processed from the profiling and attack trace files during each phase. These parameters influence both the quality of the learned LDA model and the reliability of the attack success rate evaluation.

The parameter `batch_size` specifies how many intermediate values are processed per sample batch. This value controls the trace segmentation described in subsection 3.5.4 and must match the structure of the trigger signal and number of POIs per trace.

The best-performing parameter configurations are discussed in section 4.5.

3.8.3 Batched Key Recovery Strategy

As described in subsection 3.3.2, the recovery of η must be divided into two batches. This is due to the structure of the CROSS signing algorithm: An element of the second batch $\mathbf{u}[k..n-1]$ is linked to k leakage values of `im2`; each of those depends itself on one value of the first batch $\mathbf{u}[0..k-1]$; Combining several `im2` probabilities would require combining multiple hypothesized values of $\mathbf{u}[0..k-1]$, which breaks the divide-and-conquer assumption and renders the attack computationally infeasible. To avoid this, we first recover $\eta[0..k-1]$ (batch 1), then derive the corresponding $u[0..k-1]$ values. These recovered values are then reused as known inputs for the second stage of the attack, which recovers $\eta[k..n-1]$.

The attack is executed in the following steps:

1. Load profiling traces and corresponding label data (using sample batches).
2. Perform SNR analysis on aligned trace intervals to identify relevant POIs.
3. Train LDA models using profiling traces (optionally with vertical stacking).
4. Load attack traces and apply LDA models to obtain class probability predictions for `im1`.
5. Map and combine `im1` probabilities across all traces to score each `eta[0..K-1]` hypothesis.
6. Select the highest-scoring candidate values for `eta[0..K-1]`.
7. Use recovered `eta[0..K-1]` and known `u_tilde[0..K-1]` to determine `u[0..K-1]`.
8. Use `u[0..K-1]` for the attack on batch 2.
9. Apply LDA models to obtain class probability predictions for `im2`.
10. Map and combine `im2` probabilities across all traces to score each `eta[K..N-1]` hypothesis.
11. Select the highest-scoring candidate values for `eta[K..N-1]`.

3.8.4 Details of Attack Procedure

After finding POIs and training the LDA model, we execute the attack.

In the first batch, we do not attempt to directly recover `u[0..K-1]` from the `im1` predictions. While it is possible to score `u` hypotheses individually per trace, `u` varies across rounds and traces, so their probabilities cannot be directly aggregated. Instead, we shift focus to `eta`, which is constant across all traces, and therefore suitable for cumulative likelihood scoring. This allows us to accumulate likelihoods per `eta[i]` over all attack traces (or rounds) by using the z candidate values for each `u[i]` and computing the corresponding `eta[i]` hypotheses. The mathematical structure of the recovery path, i.e., how `u`, `sigma`, and ultimately `eta` are derived, is detailed in Figure 3.2. This mapping is exploited both in the forward direction (to generate intermediates from hypotheses) and in the backward direction (to recover `eta` from `u[i]` candidates).

The highest scoring `eta[i]` is selected. Afterward, for each trace, we pick the `u[i]` that maps to the hypothesis of `eta[i]` with the highest individual likelihood.

This results in k consistent candidates for `eta[0..K-1]` and `u[0..K-1]`, which are then used to build hypotheses for the second batch.

The second batch proceeds similarly, now using the LDA model trained for `im2` values. We again use z candidate values per `u[i]`, derive the corresponding `eta[i]` hypotheses, and accumulate their likelihoods across all attack traces. Because `eta` remains constant across traces, we can again sum likelihoods for each `eta[i]` and select the most likely one.

Together with batch 1, we now obtain a full vector of highest-scoring candidates for `eta[0:N-1]`.

4 Experimental Results

4.1 Results for Simulated Attacks

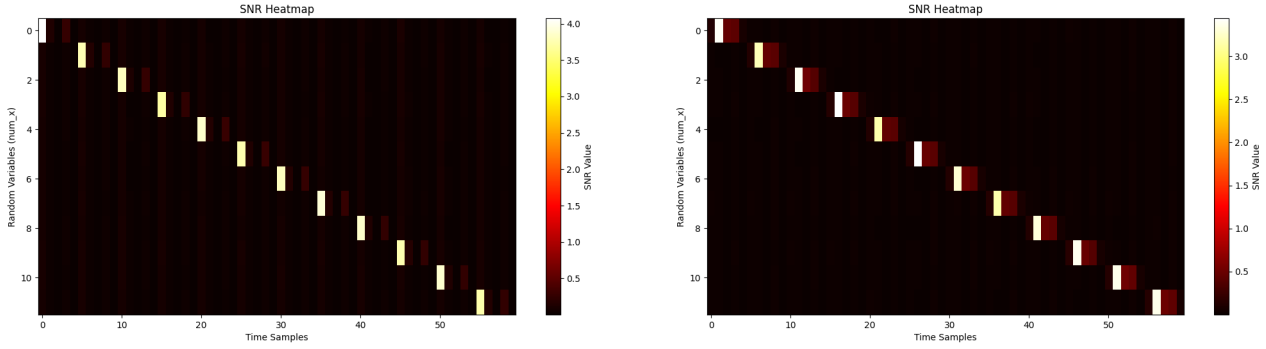
4.1.1 Overview

The tests with simulated traces were, unless otherwise stated, executed using the equivalent of a single attack trace, i.e., $w = 85$ traces with only round 0. The profiling traces were generated using the crafted inputs described in subsection 3.5.2 to support both approaches: the vertical stacking method, where the problem of missing LDA classes is mitigated by using multiple labels per column; and the regular approach with one LDA model per intermediate value. Each intermediate computation produces only five samples per inner-loop execution. Therefore, POI selection in simulated traces consistently used the "fixed" strategy with small values for `npoi`. The "centered" strategy was not used. SNR only varies with the noise standard deviation (std), so for equal noise levels, the SNR values remain consistent across runs. Table 4.1 describes and compares several test configurations and their results for SNR and LDA.

Increasing the additive Gaussian noise (`std`) leads to predictable reductions in SNR. SCALib detects very similar leakage levels for `im1` and `im2`, with slightly higher SNR observed for `im1`.

The generated traces contain exactly one leakage sample for each intermediate value. Thus, we expect to find one sample column with a distinctly higher SNR value compared to the others. The SNR results confirm this: Each simulation trace leaks five intermediate values per loop iteration, resulting in $k(n-k) \cdot 5 = 19380$ samples per trace. High SNR values occur at expected sample columns that correlate with the known intermediate values. The subsequent trace samples reflect the execution of the vector-matrix multiplication and may leak additional information. This is observed more clearly in simulations with lower noise levels. This effect is more pronounced for `im2` samples, likely due to the subsequent reduction operation: for many values, this step does not alter the Hamming weight, preserving the leakage characteristics of the addition result. With increased noise, this effect diminishes, but is more stable for `im2` than for `im1`.

The SNR heatmaps in Figure 4.1 illustrate the effects described above. The 12 consecutive intermediate values are ordered in rows from top (first intermediate) to bottom (last). We expect the peak SNR values to appear in the same order from left (earlier trace samples) to right (later samples). Each heatmap clearly shows high SNR values found for the 12 intermediates. The plots consistently reveal 12 distinct high-SNR regions, validating the effectiveness of POI selection. For `im1`, we find that always the first of five samples is dominant, whereas for `im2` the second sample consistently has the highest SNR value. For `im2` we can also see that the samples following the dominant one still have relatively high SNR values.



(a) SNR heatmap for 12 `im1` at std = 1.0

(b) SNR heatmap for 12 `im2` at std = 1.0

Figure 4.1 Comparison of SNR heatmaps for `im1` and `im2` from simulated traces at std = 1.0

Configuration				Results	
Run	*std*	vstack	npoi im1/im2	max. SNR im1/im2	LDA (A) im1/im2 (%)
1	1.0	✗	1 / 1	3.1–3.9 / 3.0–3.5	28.6 / 28.3
2	3.0	✗	1 / 1	0.41–0.44 / 0.32–0.41	11.15 / 11.10
2	3.0	✓	1 / 1	0.41–0.44 / 0.32–0.41	11.16 / 11.12
4	12.0	✗	1 / 1	0.02–0.03 / 0.02–0.03	7.92 / 7.99
5	12.0	✓	1 / 1	0.02–0.03 / 0.02–0.03	7.93 / 7.99
6	12.0	✗	2 / 2	0.02–0.03 / 0.02–0.03	7.95 / 8.05
7	12.0	✓	2 / 2	0.02–0.03 / 0.02–0.03	7.93 / 7.95
8	12.0	✗	1 / 1	0.02–0.03 / 0.02–0.03	7.93 / 7.99

Table 4.1 Intermediate Evaluation (SNR/LDA) for simulated attacks. LDA (A) shows average correct-class probabilities for im1 and im2 on attack traces.

SCALib's LDA function provides probabilities for the LDA classes for the given POI samples. The results for LDA in Table 4.1 show the averagy probabilities assigned to the correct Hamming weight classes. We can see that the LDA predictions for the correct classes are above the value we expect for random guessing, which is 7.69% due to 13 Hamming weight classes used for both types of intermediate values. For the simulated traces, the LDA results on attack traces for `im1` and `im2` are very close and drop with increased noise and lower SNR values. Although not listed, the results for LDA predictions on profiling traces closely resemble those on attack traces.

Table 4.2 shows the averages for the highest and second-highest probability for η_i candidates. Furthermore it describes the minimum, maximum and average distance between the best and second-best candidates. The recovery of η is clearly worse for batch 1 (using `im1`) than for batch 2 (with `im2`), as the average margins between the best two candidates for η_i show. Since SNR values and LDA predictions are very similar for both types of intermediate values, this is expected to be a result of two effects: the challenge with distinguishing possible `im1` hypotheses from each other as earlier mentioned in subsection 3.5.2, which will be addressed in section 4.2; furthermore, each $\mathbf{u}[\mathbf{i}]$ candidate in batch 1 has less `im1` leakage values $(n-k)$ than the $\mathbf{u}[\mathbf{i}]$ candidates in batch 2 with k `im2` leakage values.

Run	Batch 1		Batch 2	
	Rel. Diff (%)	Top/2nd (%)	Rel. Diff (%)	Top/2nd (%)
1	12.9 / 24.0 / 19.5	28.3 / 22.8	37.7 / 41.0 / 39.6	28.0 / 16.9
2	3.53 / 7.90 / 5.88	10.97 / 10.32	9.24 / 11.68 / 10.86	10.98 / 9.79
3	3.83 / 7.95 / 5.83	10.99 / 10.35	9.64 / 11.55 / 10.63	11.01 / 9.84
4	0.11 / 1.05 / 0.59	7.86 / 7.81	0.52 / 1.72 / 1.09	7.92 / 7.83
5	0.09 / 1.06 / 0.62	7.84 / 7.79	0.69 / 1.41 / 1.04	7.82 / 7.74
6	0.06 / 1.09 / 0.57	7.87 / 7.83	0.90 / 1.68 / 1.31	7.98 / 7.88
7	0.10 / 1.18 / 0.64	7.85 / 7.81	0.87 / 1.65 / 1.30	7.88 / 7.77
8	0.25 / 1.10 / 0.62	7.86 / 7.81	0.64 / 1.33 / 1.00	7.91 / 7.83

Table 4.2 η recovery metrics per batch (simulated traces, with one attack trace). "Rel. Diff" shows the minimum, maximum, and average relative differences between the top two candidates. "Top/2nd" lists the average probabilities of the best and second-best candidate.

In all shown attack scenarios with simulated traces, η was fully recovered. Since η and $\mathbf{u}$ in our tests are throughout completely recovered for batch 1, we always work with fully correct batch 1 hypotheses for batch 2. If batch 1 cannot be recovered completely, the results for batch 2 would be affected negatively due to missing or wrong hypotheses in batch 1. Despite narrow candidate margins, the correct η was consistently selected, suggesting the attack's robustness even with small threshold values.

4.1.2 Influence of Attack Parameters

This section will evaluate how key attack parameters impact the attack success and prediction accuracy. Comparative results from parameter sweeps will be summarized in a dedicated results table.

std. Full key recovery remained possible even at `std` = 12.0. These results confirm that, under idealized noise conditions, our LDA-based approach achieves correct results even in the presence of strong noise, providing a useful reference for evaluating real-trace scenarios.

npoi. We created one leakage sample per intermediate value in the simulated traces. Also the values following each intermediate value within an inner-loop execution are calculated as we expect them in a binary built from the C reference implementation. Although SNR plots for each intermediate typically show one sample with a clearly highest SNR value, using $npoi = 2$ (see subsection 3.8.2) improves recovery accuracy compared to $npoi = 1$, especially for the second batch of key values. This indicates that even samples with lower SNR contribute meaningful leakage when aggregated, especially for `im2`, where the reduction structure retains Hamming weight information. For `im1`, the impact is less pronounced, which aligns with earlier observations that recovery is more difficult in the first batch. Accordingly, the enhancement for `im2` and batch 2 does not improve the attack significantly.

vstack. With little profiling traces (e.g. 200, i.e., not all crafted inputs are utilized), *vertical stacking* (`vstack=1`) (explained in subsection 3.5.3) leads to a clear improvement. In such cases, it provides the LDA model with enough training data in the first place, whereas having many LDA models with little training data is not sufficient and results in *missing classes* for the LDA model. For simulated traces, this vertical stacking approach works well: the alignment of POI sets is flawless, since those have exactly five intermediates per inner-loop execution, and the correct leakage samples (depending on setting of `npoi`: one or two) are clearly identified as POIs.

Using the whole set of crafted traces, we can omit *vstacking* because all classes are made available for all intermediate values and POI sets. The amount of profiling data which can improve the LDA models seems to be exhausted with those, and *vstacking* does not yield any additional advantage.

Sample batch size. Since simulated traces contain substantially less data than captured traces, the sample-batch approach mentioned in subsection 3.5.2 is not necessary for those. To test that the approach using trace sample-batches works in principle and that the implementation is correct, we tested with different batch sizes (from 12 POI sets per batch to the full $k(n-k) = 3876$ POI sets per batch). As expected, the sample batch size did not have any influence on the LDA and η recovery results.

LDA_p. The dimensionality of the LDA projection space is outlined in subsection 3.8.2. In the simulation, its effect is limited due to the small number of input dimensions (at most two POI samples per intermediate). We observed no notable difference when varying `LDA_p`, as the original feature space was already low-dimensional. A more meaningful evaluation will be possible in the context of real traces with larger POI windows.

Number of attack and profiling traces. The attack on simulated worked with as little as 200 profiling and 20 attack traces (all with round 0 only, i.e. less than 25% of a full signature trace with $w = 85$ rounds) when utilizing *vertical stacking*. A lack of attack traces obviously impedes the attack.

Beyond using one full attack trace (i.e. $w = 85$ traces with round 0), the efficiency of the attack appears to plateau. Even when using 10 full traces (i.e., 10×85 round-0 segments), the margins between the top two η_i candidates remained small for many indices. We attribute this to insufficient distinguishability between competing `im1` hypotheses: section 4.2 describes that many values produce similar or identical Hamming weights under most combinations.

Threshold. To assess the robustness of our predictions, we also tested threshold-based acceptance: we required a minimum confidence margin between the top and second-best η_i candidate, rejecting predictions that failed to meet it. However, even when using 10 full attack traces (10×85 individual round-0 traces), the confidence margins remained low in many cases. This prevented practical use of thresholding for η_i recovery, especially at typical cutoffs (e.g., 20%). This suggests that while the attack is robust, its confidence margins are inherently limited unless the trace model or scoring function is further improved. As a result, we default to always selecting the top candidate for each index, even at the cost of occasional misclassifications.

4.2 Evaluation of Intermediate Value Separation

During the simulation phase, we observed that while it is possible to reduce the hypothesis space for values of `u[i]` from p hypotheses per value to z hypotheses, those that remain are often difficult to distinguish when targeting *intermediate 1* values as results of multiplication.

The calculation of $\mathbf{u}$ uses the vector product implementation as described in Listing 4.1. The function uses `FQRED_DOUBLE` as described in paragraph 3.3.2.

```
1 void fq_vec_by_fq_vec_pointwise(uint8_t res[N],
2                                 const uint8_t in1[N],
3                                 const uint8_t in2[N]){
4     for(int i = 0; i < N; i++){
5         res[i] = FQRED_DOUBLE( (uint16_t) in1[i] *
6                                (uint16_t) in2[i] );
7     }
8 }
```

Listing 4.1 Component-wise vector multiplication with lazy reduction in CROSS

When we examine the resulting values for $\mathbf{u}$, we see that in all *CROSS-R-SDP* variants, the values of σ in $\{0..z-1\} = \{0..6\}$ are transformed into values for $\mathbf{v}[\mathbf{i}] \in \{1, 2, 4, 8, 16, 32, 64\}$, i.e., powers of 2: $\mathbf{v}[\mathbf{i}] \in \{2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6\}$. When used in multiplications, these values either do not change a value or its Hamming weight (when multiplied by $v = 1$), or they apply a left-shift to the binary representation, which typically preserves the Hamming weight for $v \in \{2, 4, \ldots, 64\}$.

We analyze the behavior of the $z = 7$ hypotheses derived from a known $\mathbf{u}' \in \mathbb{F}_p^n$. We use $\mathbf{v} \in \{1, 2, 4, \ldots, 64\}^n$ to denote vectors of scalar multipliers (i.e., powers of two), derived from the z values of σ.

Given: $\mathbf{u}' \in \mathbb{F}_p^n, \quad \mathbf{v}^{(j)} = 2^j \cdot \mathbf{1}, \quad j \in \{0, \ldots, 6\}$

Hypotheses: $\mathbf{u}^{(j)} = \texttt{FQRED\_DOUBLE}(\mathbf{v}^{(j)} \star \mathbf{u}'), \quad j = 0, \ldots, 6$

Each $\mathbf{u}^{(j)}$ has the same Hamming weight as $\mathbf{u}'$ (no change due to shift or lazy reduction)

Derived hypotheses: $\texttt{im1}^{(j)} = \mathbf{u}^{(j)} \star \mathbf{v}_{\text{tr}}, \quad j = 0, \ldots, 6$

Using a known value of $\mathbf{u}'$ to calculate the z possible values for the corresponding element of $\mathbf{u}$ via component-wise multiplication and the *lazy reduction* described in paragraph 3.3.2, we find that the resulting z hypotheses of $\mathbf{u}$ always have the same Hamming weight, as shown in tables 4.3 and 4.4.

Since `im1` values are also the results of multiplications with these values of $\mathbf{u}$, many `im1` values are also affected. Thus, the distinguishability of the different hypotheses is impeded by this effect. The Hamming weight for derived `im1` hypotheses can differ across the z `im1` hypotheses if the calculated `u` hypothesis required reduction, as in Table 4.3. However, as shown in Table 4.4, this is not always the case.

v	$u = \texttt{FQRED\_DOUBLE}(u\_\text{tilde} \cdot v)$	HW(u)	$im1 = u \cdot v\_\text{tr}$	HW($im1$)
1	126	6	13482	7
2	125	6	13375	9
4	123	6	13161	8
8	119	6	12733	9
16	111	6	11877	8
32	95	6	10165	9
64	63	6	6741	7

Table 4.3 Example for im1 Hamming weights: $u\_\text{tilde} = 126$, $v\_\text{tr} = 107$

v	$u = \texttt{FQRED\_DOUBLE}(u\_\text{tilde} \cdot v)$	HW(u)	$im1 = u \cdot v\_\text{tr}$	HW($im1$)
1	3	2	237	6
2	6	2	474	6
4	12	2	948	6
8	24	2	1896	6
16	48	2	3792	6
32	96	2	7584	6
64	65	2	5135	6

Table 4.4 Example for im1 Hamming weights: $u\_\text{tilde} = 3$, $v\_\text{tr} = 79$

It is important to note that this limitation is particularly relevant for `im1`, which is directly influenced by `u`. In contrast, `im2`, which results from an addition, appears to exhibit greater variability. However, a full analysis that could confirm better distinguishability of `im2` is left for future work.

Figure 4.2 shows an analysis evaluating all possible combinations of $u' \in \mathbb{F}_{127}$ and $v_{\text{tr}} \in \mathbb{F}_{127}$, totaling $127 \cdot 127 = 16129$ cases. For each pair, we compute the seven hypotheses for $u = \texttt{FQRED\_DOUBLE}(u' \cdot v)$ with $v \in \{1, 2, \ldots, 64\}$, and then derive the corresponding $\texttt{im1} = u \cdot v_{\text{tr}}$. The value D indicates the number of distinct Hamming weights among the seven `im1` hypotheses. A value of $D = 1$ means all seven hypotheses yield the same Hamming weight, making them indistinguishable based on that leakage. Higher values of D indicate more variation among the hypotheses' Hamming weights, hence better distinguishability. In CROSS-R-SDP, `im1` values can take 13 Hamming weight classes.

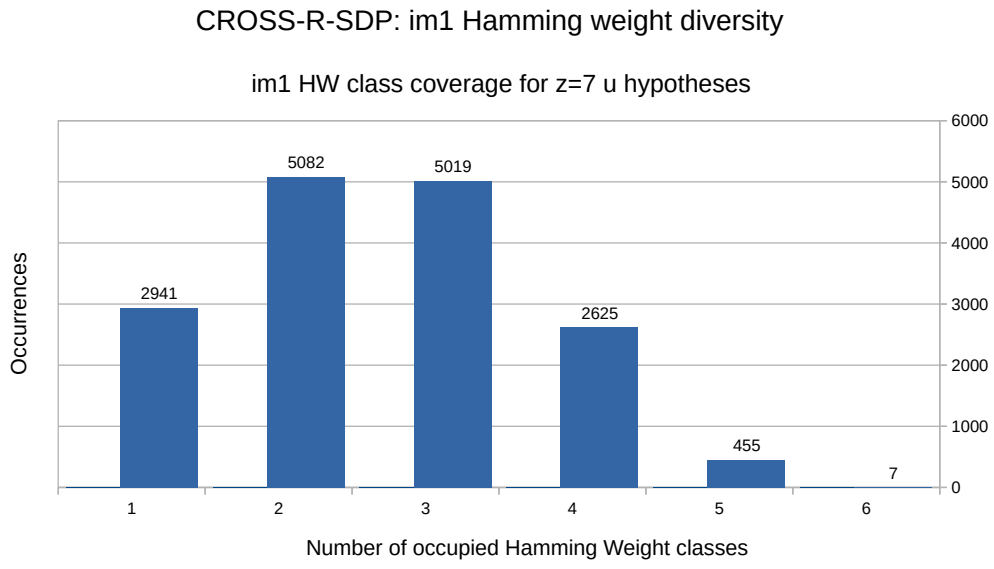


Figure 4.2 im1 Hamming weight distribution

In the majority of cases, the results fall into 2 or 3 HW classes. Thus, we obtain the same HW classes for different hypotheses of u[i], which makes it difficult to identify the correct hypothesis unless sufficient leakage values are available.

In summary, although the z-to-HW ambiguity can sometimes be reduced through the multiplication with v_{tr}, in most cases the resulting `im1` hypotheses still yield only 2 or 3 distinct HW classes. This limits the effectiveness of side-channel classification and constrains recovery, especially for indices η_i where multiple candidates remain ambiguous.

4.3 Evaluation of Trace Quality and Leakage

4.3.1 Interval Lengths and Alignment

A visual inspection of trace plots such as Figure 4.3 and Figure 4.4 confirms the plausibility and correctness of the acquired traces. As described in Listing 3.1, we set the high trigger at the begin of the vector-matrix multiplication (i.e. begin of inner loop execution) and reset it before leaving the inner loop. The plots show excerpts of samples and triggers of three traces with a downsampling factor of 10 as overlay.

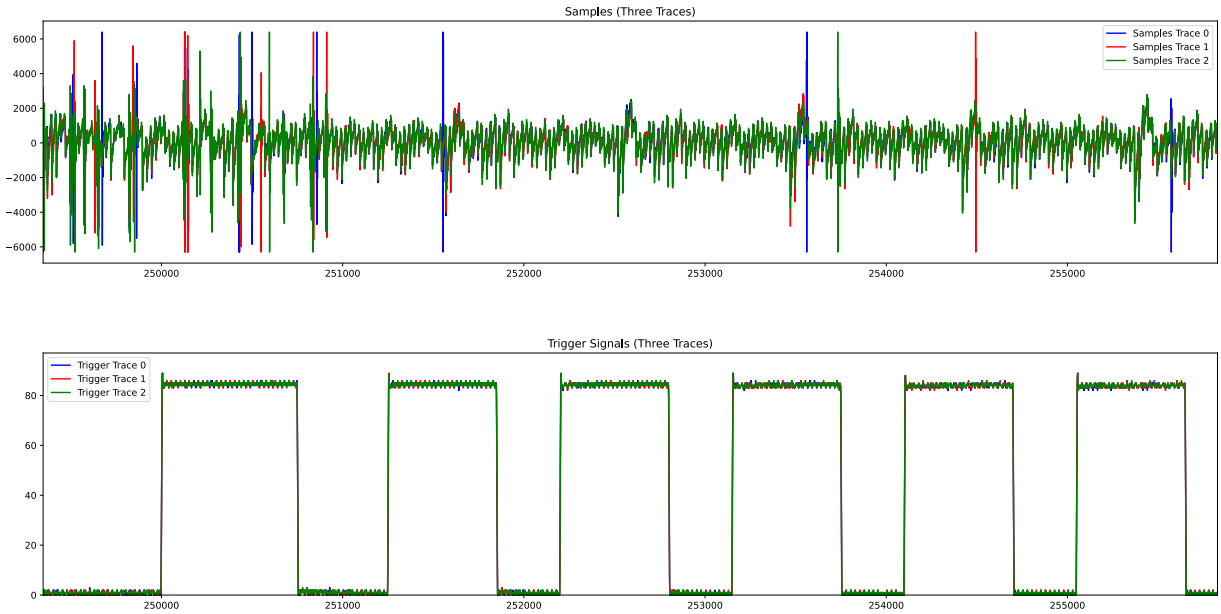


Figure 4.3 Samples and triggers for three traces, dsf=10 (begin of execution)

In Figure 4.3 we can clearly identify six segments with a distinctive high trigger, each corresponding to one execution of the inner loop. The triggers for the three traces are well aligned.

The first high trigger segment in each trace is consistently longer than the following ones. This is likely caused by microcontroller internals such as cache misses or one-time memory fetches for operands (e.g., `V_tr` or `u`) during the first execution. The remaining trigger intervals reflect regular inner-loop executions with minimal variation.

Figure 4.4 shows three low trigger segments, of which the middle low interval is longer. This occurs if the inner loop terminates and the outer loop is executed. It is the same for all traces and does not have negative effects on the alignment of traces. The stability of the trigger alignments was also verified using a script to identify the high trigger intervals for different measurement under the same conditions, as previously described in subsection 3.7.3.

The SNR computation is agnostic rearding such irregularities. For the training of the template model via LDA, the regularity of sample positions is also not of importance—the LDA only receives the

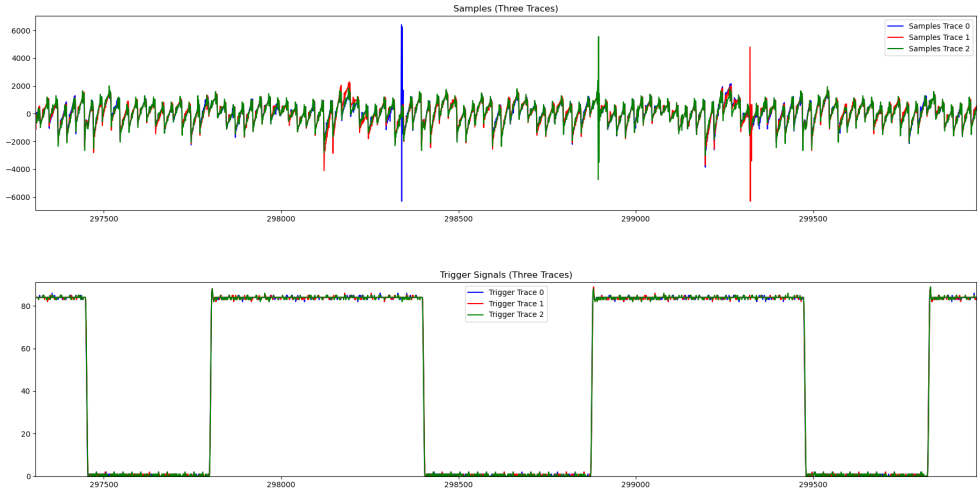


Figure 4.4 Samples and triggers for three traces, dsf=10 (with outer loop)

sample values and is agnostic of their positions within a trace. In particular, also the vertical stacking of POIs is not affected in any way, because it stacks sample sets of the same size `npoi` irrespective of their previous positions in a trace. For prediction probabilities using attack traces, it is important that the POIs of the profiling traces match the sample positions in the attack traces.

The intervals have roughly lengths of 5950 samples, which is at 250 samples per clock cycle approx. 24 clock cycles. An example high triggers shows a length range of 5949–5960 samples for the high intervals (except for the first interval). Figure 4.5 shows an examplatory length distribution for a measured trace. Executions of the inner loop are identified by locating high trigger segments of at least 50 samples and aligning them with 250-sample boundaries (matching the known sampling rate per cycle).

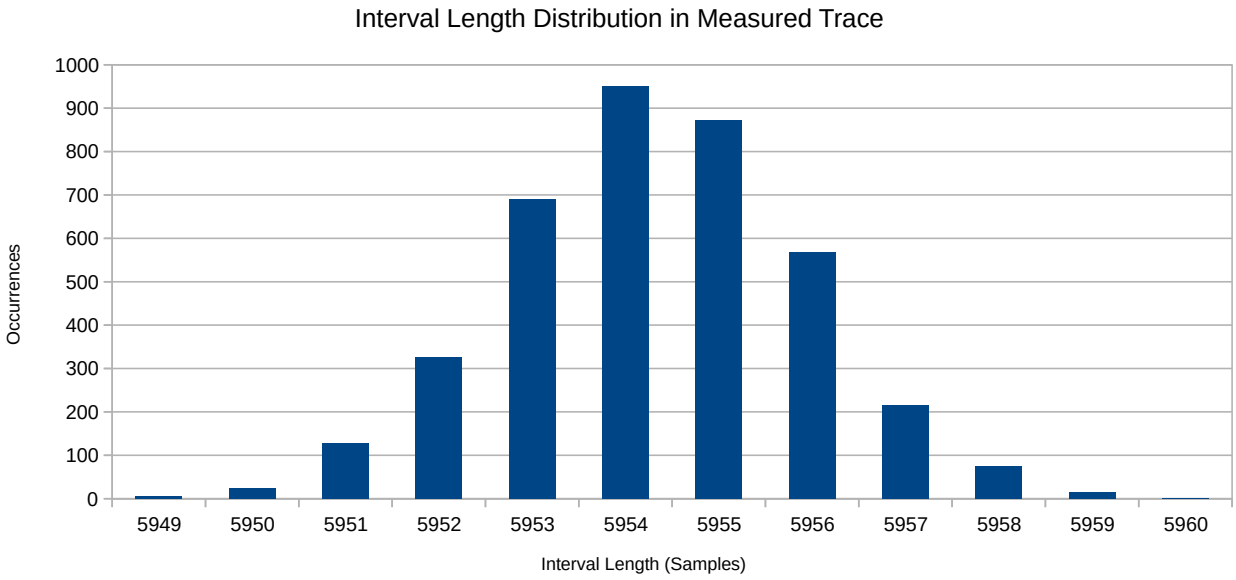


Figure 4.5 Interval Length Distribution

For 3876 executions with high triggers, we have 3875 low trigger intervals inbetween those. Our script consistently identifies 3799 short distances (low trigger intervals) between high trigger intervals (only inner loop execution) with distances of 9493–9507 samples, and 75 intervals with longer low trigger

segments in a slightly elevated range(10745–10754 samples). The first interval is again an outlier with an even higher distance. These correspond well with the 76 outer loop executions.

4.3.2 Visual Inspection of POI Power Samples

The DSO records traces at 8-bit resolution, meaning each sample can distinguish between 256 discrete voltage levels within the configured input range. Figure 4.6 shows an excerpt from a power trace using a downsampling factor of 10, i.e., 25 samples span one target clock cycle. The traces comprise approximately 200 samples and highlight POI regions for `im1` and `im2`. In this example, traces 0 and 1 were generated using identical inputs (via deterministic execution with a CSPRNG), ensuring they process the same internal values. Trace 2, in contrast, uses different inputs for comparison. The power samples from three traces generally follow a consistent low-frequency pattern, indicating good alignment across captures. This shape is often dominated by repeated instructions and memory accesses (e.g., register loads, stack operations) that produce similar leakage across traces.

Despite the overall similarity, small amplitude differences between traces are clearly visible. These trace-to-trace deviations are likely caused by value-dependent leakage and operand variability during the relevant instruction cycles.

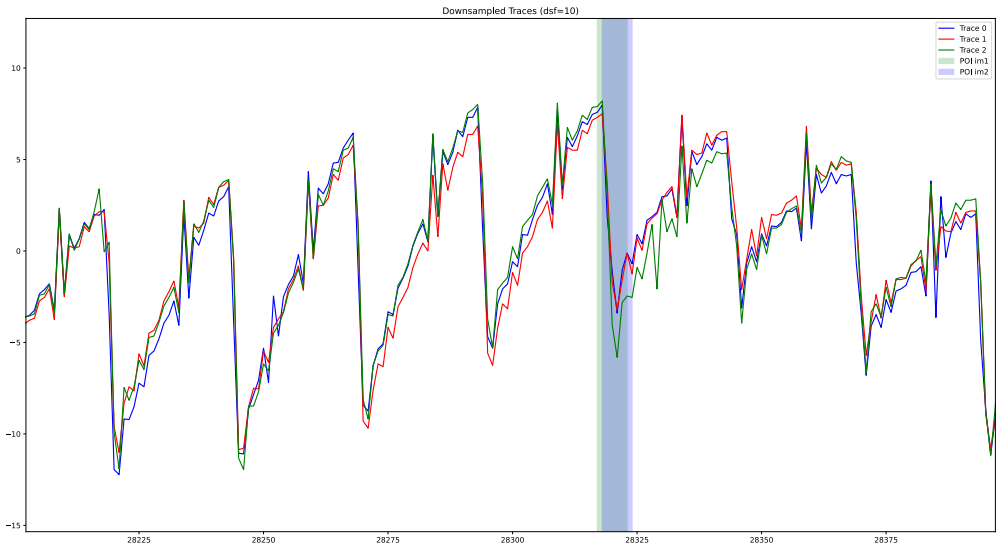


Figure 4.6 Trace excerpt with approx. 200 power samples (dsf=10)

For both `im1` and `im2`, the sets of selected points of interest (POIs) were largely overlapping. This suggests that the underlying instructions responsible for leakage are executed in a similar temporal region. The higher SNR values observed for `im2` suggest that the corresponding instructions may produce a more reliable signature when monitored, compared to those linked with `im1`.

The visual similarity between trace segments near the POIs does not guarantee identical leakage characteristics. To further investigate, we analyze the model predictions for selected traces in subsection 4.4.2, including their actual labels and LDA-derived class probabilities. This detailed inspection reveals both the strengths and limitations of the LDA model in capturing subtle, value-dependent leakage effects.

4.3.3 Evaluation of SNR Values

While for simulated traces, the SNR values were determined by the level of added Gaussian noise, the noise in captured traces is determined by internals of the MCU. For traces obtained via measurements, the SNR values vary with the preprocessing of the traces. As described in subsection 3.7.4, we

downsample the obtained traces. We experimented with different downsampling parameters, and identified the values 25, 50, and 125 to yield the best results for both types of intermediate values (Table 4.5). The measured SNR values for real traces are significantly lower than those obtained from simulated traces, even when using a noise standard deviation of 3.0, the maximum noise authors measured in [9] for an almost identical device. This discrepancy highlights a key limitation of simulations: they struggle to accurately model the complex and often device-specific leakage behavior observed in real-world hardware.

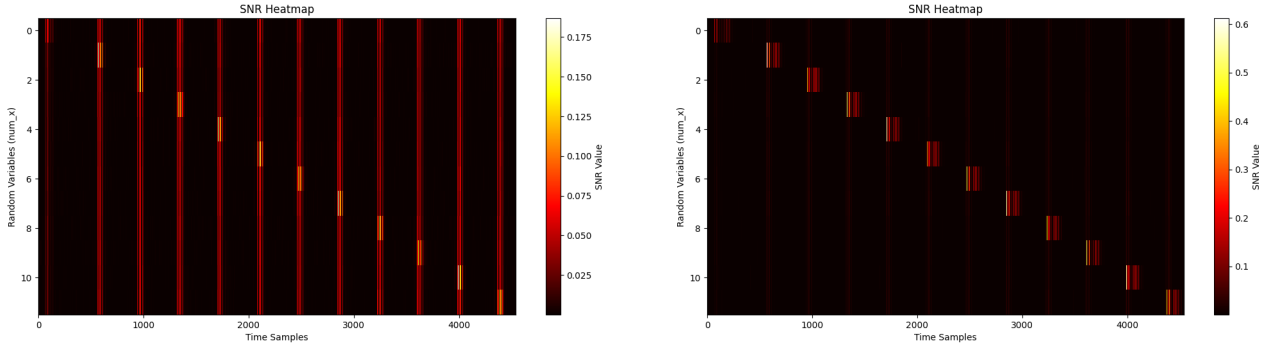
dsf	SNR max (im1)	SNR max (im2)
1	0.06–0.12	0.15–0.38
5	0.05–0.10	0.17–0.40
10	0.05–0.11	0.15–0.35
25	0.15–0.18	0.51–0.59
50	0.12–0.20	0.58–0.65
125	0.14–0.18	0.30–0.60
250	0.13–0.15	0.30–0.35

Table 4.5 Typical maximum SNR values for im1 and im2 across varying downsampling factors (dsf). Note: SNR evaluations are based on a profiling set of 20,000 measured traces.

We initially suspected that the use of a fused multiply-accumulate (MLA) instruction in the optimized binary (`-Os`) may cause low leakage and thus very low SNR values. Interestingly, varying compiler optimizations (e.g. `-O0`, which uses separate instructions for multiplication and addition) did not significantly affect SNR values in our setup. This suggests that the observed leakage pattern is determined by the microarchitectural implementation of the ARM Cortex-M4 CPU, and that the MLA instruction behaves similarly to individual scalar operations in terms of power leakage.

Since SNR values indicate how much data-dependent information is present in a signal, we expect better attack results with configurations that yield higher SNR. We therefore focus our attack evaluation on traces downsampled with factors 25, 50, and 125.

A particularly interesting difference between the SNR values for simulated and real traces is the deviation between `im1` and `im2`. We see that the results for the multiplication, i.e. results for `im1`, yield much lower SNR values than for `im2`. The different operations (multiplication, addition) themselves seem not to be the cause of this, because this should be visible in simulated traces as well.



(a) SNR heatmap for 12 `im1` at dsf = 25

(b) SNR heatmap for 12 `im2` at dsf = 25

Figure 4.7 Comparison of SNR heatmaps for `im1` and `im2` from real traces with dsf = 25

While both intermediates are computed using known and fixed values, `im2` typically results from an arithmetic combination involving a 16-bit operand. This may trigger more complex data transfers and register usage inside the microcontroller, creating stronger or more structured leakage. In contrast, `im1` results from a simpler multiplication between small field elements, possibly involving optimized or unrolled instructions with lower signal variation.

The heatmaps in Figure 4.7 highlight another difference between `im1` SNR values for captured and simulated traces: As in the previous heatmaps, 12 neighboring intermediate values are ordered in rows from top to bottom. We find that relatively high SNR values for each `im1[i=I]` value are also detected in other trace segments than the expected ones. This is indicated by the red lines in each row, in locations where other `im1` have their peak SNR values (yellow/orange).

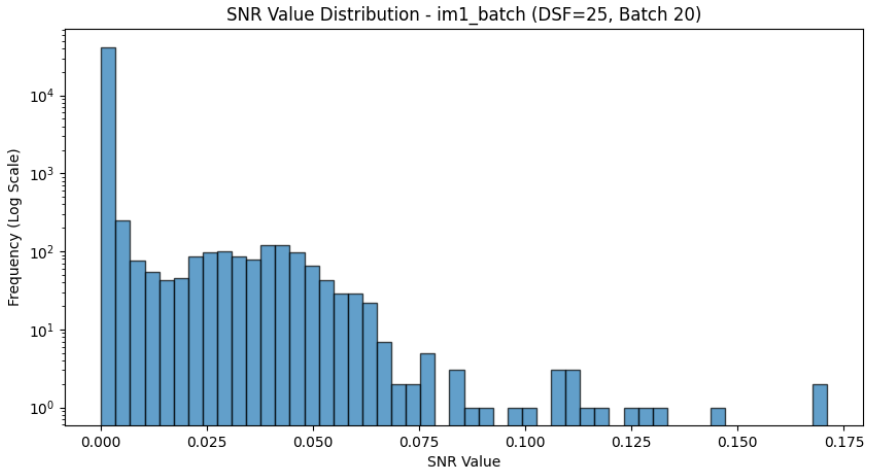


Figure 4.8 Distribution of SNR values for one `im1` with dsf = 25

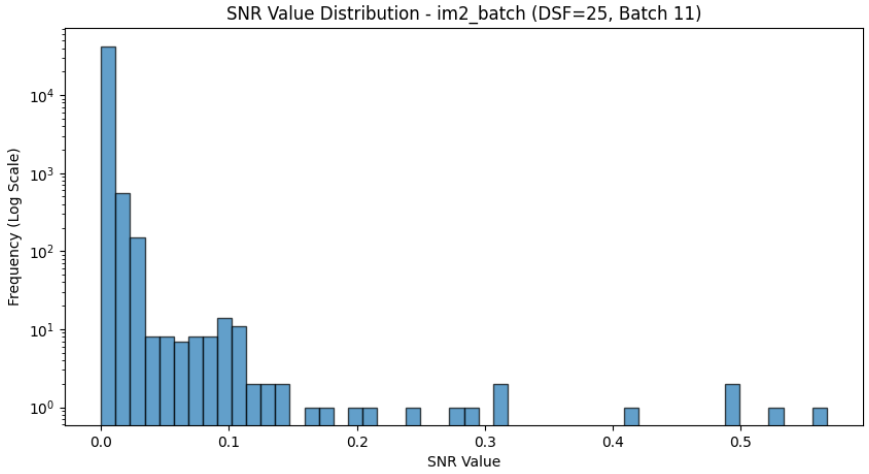


Figure 4.9 Distribution of SNR values for one `im2` with dsf = 25

The plots in figures 4.8 and 4.9 confirm this. Each of them shows the distribution of SNR values for one intermediate in one of 34 trace segments (limited sample-batch, as described in subsection 3.5.4) of $3876/34 = 114$ expected leakage areas for POIs. We can see a more distinctive separation between lower and higher SNR values for `im2`. The distribution for `im1` shows few outstanding high SNR values, but also many mid-range SNR values.

While figures 4.10 and 4.11 are confirming those results, they also show that the peak SNR values for `im1` values are found reliably at the expected positions. This insight can support parameter selection for efficient attacks. In particular, the combination of downsampling factor and `npoi` must strike a balance: enough temporal resolution to preserve peak SNR regions, but aggressive enough to reduce feature count for reliable LDA training.

These results also show the limitations of simulated traces; those do not contain unintended or secondary leakage and have no crosstalk across variables unless explicitly modeled.

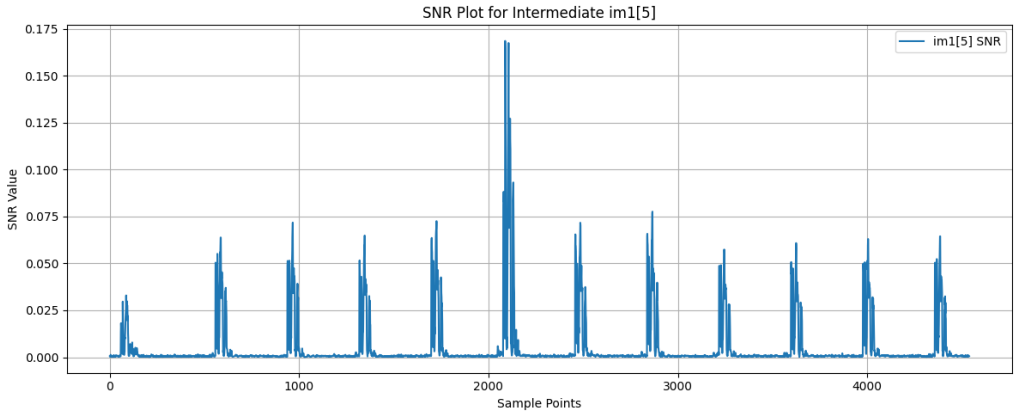


Figure 4.10 SNR peak values for one `im1`

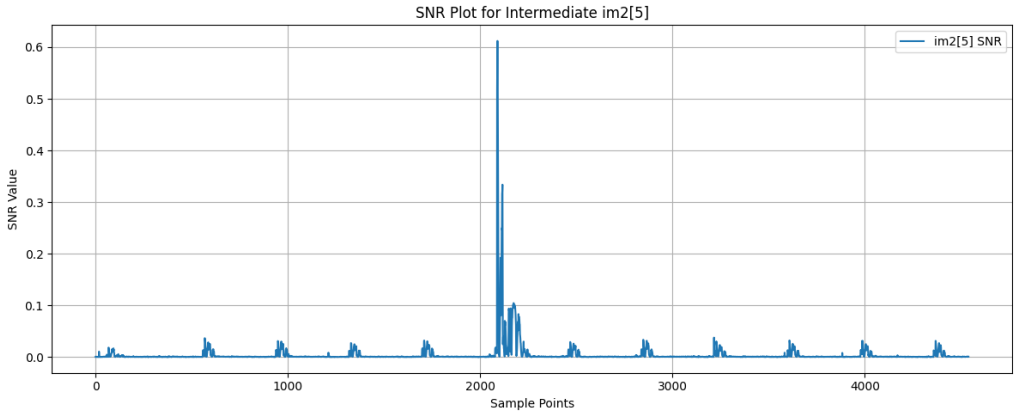


Figure 4.11 SNR peak values for one `im2`

4.3.4 First Rounds with $b = 1$

An unexpected but consistent pattern emerged during trace inspection: the number of traces where round 0 uses $b = 1$ is significantly lower than expected. Given our scenario with $t = 163$ rounds and $w = 85$, more than half of the rounds should involve $b = 1$. Accordingly, we expected about 50% of traces to exhibit $b = 1$ in round 0, assuming uniform randomness in challenge selection.

However, empirical results across multiple attack trace files showed only around 30% with $b = 1$ in round 0:

- 1000-trace file (round 0 only): 337 traces with $b = 1$
- 200-trace file: 68 traces with $b = 1$
- 3000-trace file: 990 traces with $b = 1$

This discrepancy remains unexplained, but raises questions about the randomness or potential bias in the challenge sampling process. The CROSS specification assumes uniformly distributed challenge vectors, so this skew could suggest a systematic effect. It may be of interest to the authors of CROSS to examine whether this bias is inherent to the scheme's challenge generation or specific to our instantiation.

4.4 Evaluation of LDA Model Behavior

4.4.1 LDA Prediction Results

Table 4.6 lists several tests with different configurations. LDA predictions were performed on a set of 3,000 attack traces (round 0) to provide reasonable statistical significance. The LDA predictions are

often in the same range as simulation traces with std 12.0, i.e., while not much above average guessing (7.69%), they manage to extract information about the processed values.

For tests with vertical stacking, increasing the number of profiling traces (ntraces_p) from 5,000 to 20,000 improved results visibly (see tests 1 and 2).

In our experiments, tests with no vertical stacking and otherwise unchanged parameters slightly improved classification results compared to no vertical stacking (see tests runs 2 and 5). This suggests that vertical stacking introduces subtle side effects that reduce model quality (e.g., alignment errors across vertically stacked POIs). Or that there are differences between the rounds that we do not factor in, and the different POI sets for different intermediates values differ quality-wise and LDA models should be trained separately for different POI of the same type.

The POI selection strategy *fixed* lead throughout to worse predictions than the same test configurations with *centered* POI selection (e.g., test 6).

Tests with a downsampling factors (dsf) of 10, 25, 50 show minor differences, but it is difficult to isolate the influence of this parameter because it is closely related to the number of POIs and the LDA dimensionality reduction `LDA_p`. The factor 125, despite its high SNR values, resulted in `im1` predictions well below for confgurations with smaller values of `dsf`, which may be related to the usage of less POI for high `dsf` (test run 7). Due to the substantial runtime of tests, we did not find time to explore more variations of `npoi` and `dsf`, which seems a crucial factor to find the best configuration.

Configuration							**LDA Results**
Run	POI	vstack	npoi	LDA p	dsf	ntraces_p	im1 / im2 (%)
1	centered	✓	7 / 7	3	25	5,000	8.35 / 12.48
2	centered	✓	7 / 7	3	25	20,000	9.24 / 12.60
3	centered	✓	15 / 15	3	10	10,000	9.19 / 8.67
4	centered	✓	5 / 5	3	50	20,000	9.29 / 12.9
5	centered	✗	7 / 7	3	25	20,000	9.36 / 13.44
6	fixed	✓	7 / 7	3	25	10,000	7.80 / 12.63
7	centered	✓	2 / 2	2	125	20,000	8.00 / 12.29

Table 4.6 Intermediate Evaluation (LDA) for attacks on real traces. Configurations include POI strategy, vstacking, LDA p, number of POIs per intermediate and number of profiling traces. Results show average correct-class probabilities from LDA.

Influence of Sample Batch Size. As discussed in subsection 3.5.4, we use vertical batching to handle the large memory footprint of captured traces. To assess whether sample batch size affects LDA model training or attack results, we ran a controlled comparison using several configurations. In all cases, the only difference was the sample batch size (ranging from 12 to 228); all other parameters were fixed.

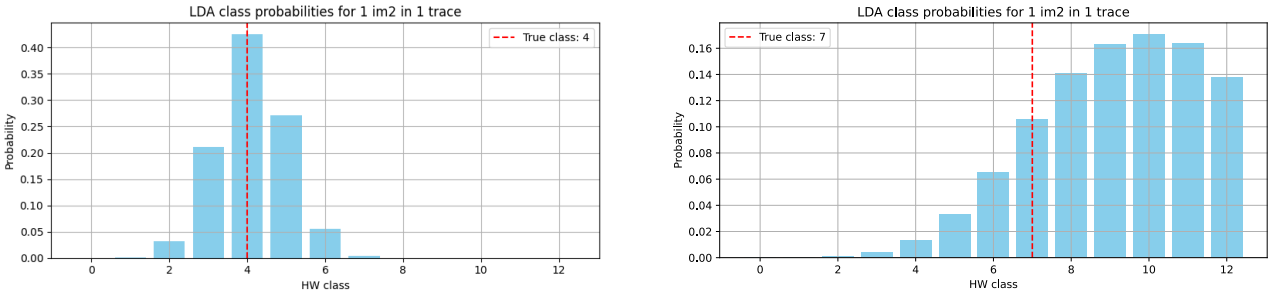
Early experiments showed variations in LDA prediction quality when using smaller batch sizes. However, these were conducted with an incomplete version of the attack script.

In later tests using the final implementation, no significant difference was observed: the predicted classes (and also secret recovery success, discussed later) were consistent across all batch sizes.

While attack results were consistent across batch sizes, runtime differed significantly. Smaller batches (e.g., 12 intermediates per batch) caused a noticeable increase in processing time, especially during preprocessing of the profiling traces. Larger batches (e.g., 114 intermediates) allow for more efficient data handling and reduce redundant preprocessing operations. Therefore, we recommend using the largest batch size that fits within memory constraints.

LDA Predictions for Individual Samples. To illustrate individual prediction behavior, each plot in Figure 4.12 shows the LDA output probabilities for a single trace and a single POI set. For the simulated trace with low noise ($std = 1.0$), the true Hamming weight is correctly assigned the highest probability, and nearby classes receive notable probability mass. Even with LDA predictions for

intermediate 2 values in general being more precise than for `im1`, the prediction for the measured `im2` sample is very unprecise in this case. Notably, the HW classes 11 and 12 get assigned rather high likelihoods, despite being very rare and in this case also wrong. Additional LDA prediction plots for other traces and configurations are included in section A.1.



(a) Simulated trace with low noise ($std = 1.0$)

(b) Captured trace with dsf=25

Figure 4.12 LDA prediction probability distributions for a single `im2` sample (single POI set and trace). The correct HW is not ranked highest in measured trace, and very rare HW values receive large probability mass. See section A.1 for more examples.

Detailed trace excerpt. We manually inspected the LDA predictions for selected trace segments. In particular, we evaluated three traces using the same POI sets chosen during training (see also subsection 4.3.2). Figure 4.13 shows the same traces more closely around the POI regions, with approximately 20 samples each. The x-axis shows the sample indices, while the y-axis shows the corresponding voltage levels in millivolts.

A downsampling factor of $dsf = 10$ results in only 25 samples per clock cycle (instead of 250), and the POI sets cover seven of these samples ($npoi = 7$). Traces 0 and 1 shared identical intermediate values, while trace 2 differed in both input and leakage. While traces 0 and 1 are strongly aligned, trace 2 slightly deviates from their curve.

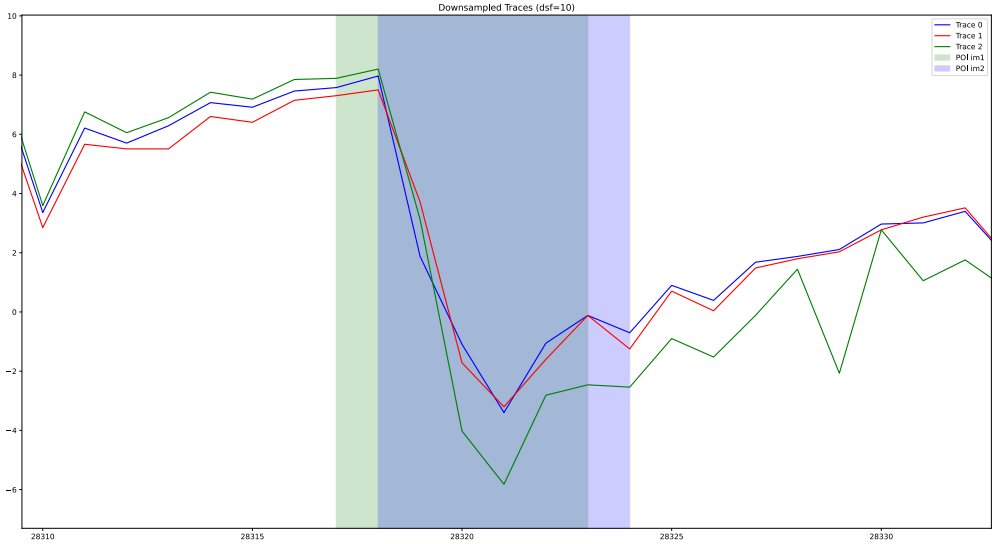


Figure 4.13 Trace excerpt with approx. 20 power samples ($dsf = 10$). POI regions are highlighted for `im1` and `im2`. Traces 0 and 1 show nearly identical leakage due to matching intermediate values. Trace 2 deviates, reflecting its different input.

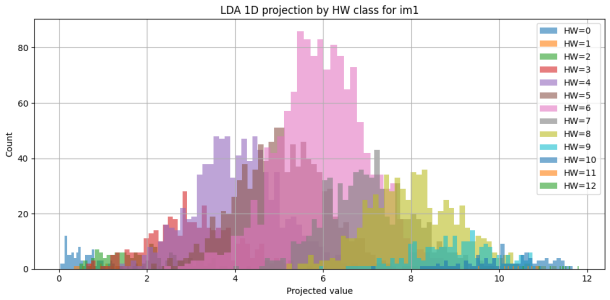
LDA predictions on selected traces. We selected the same three traces and evaluated LDA predictions for both `im1` and `im2` intermediates at a single POI set. Traces 0 and 1, which share identical intermediate values, yield nearly identical predictions. However, for trace 2—despite having a correct Hamming weight of 3 or 4—the LDA model assigned significant probability to implausible classes such as HW = 0 or HW = 1. This behavior, observed for both intermediates, reveals that the model may be confidently wrong in individual cases. We show selected prediction plots in Figure 4.12 and include further examples in section A.2.

This highlights a critical limitation: even when the model is accurate on average, its top predictions for specific inputs may be dominated by statistical outliers or noise artifacts. This supports the need for margin-aware recovery methods that consider multiple candidates rather than relying solely on top-1 classification.

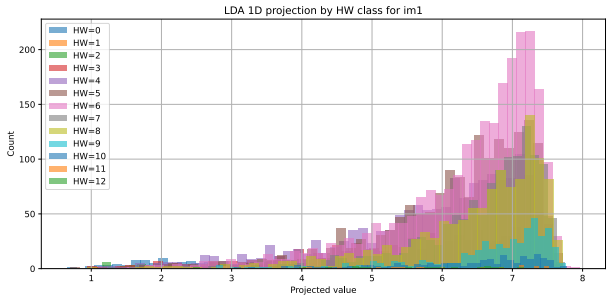
4.4.2 Statistical Analysis of LDA Output Distributions

To better understand the average behavior of the trained LDA model, we analyzed its class probability outputs across all traces and POI sets.

Score distributions across classes. We next evaluated how the LDA model distributes probability across Hamming weights for different actual values. Figure 4.14 shows the average predicted class distribution for each true Hamming weight, over a single POI set. Ideally, each HW class would yield a narrow, centered probability mass. For simulated traces (low noise), this behavior is observed. For measured traces, the projection becomes noisier and less well separated. More such plots are shown in section A.2.



(a) Simulated trace (std=1.0): LDA projection of probabilities per actual HW class



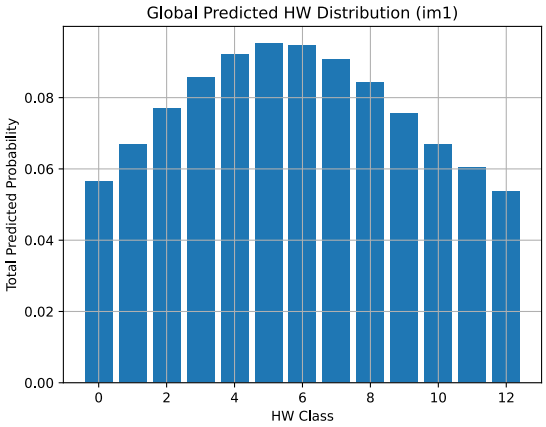
(b) Measured trace (dsf=25): LDA projection of probabilities per actual HW class

Figure 4.14 LDA probability projections for each actual HW class over a single POI set (intermediate: `im1`). Simulated traces show centered distributions, while measured traces show higher variance and confidence in implausible classes.

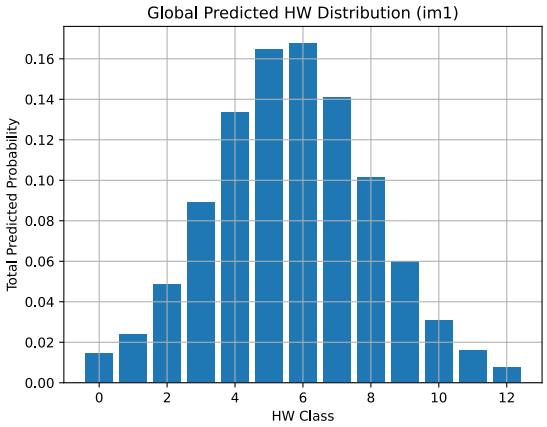
Global HW prediction distributions. In subsection 3.5.2 we discussed the expected and empirical distributions of `im1` and `im2` Hamming weights. These can now be compared with the global HW class predictions made by the LDA model (across all POI sets and traces). Figure 4.15 shows the results for `im1` under two noise conditions. We observe that rare classes (e.g., HW $= 0, 1, 12$) are assigned disproportionately high probability mass—even though their actual occurrence is rare. This further confirms the model's tendency to misclassify toward statistical edges. section A.2 includes similar results for `im2` and captured traces.

The LDA model works as expected for simulated traces under low-noise conditions; confirming the model's validity and correct application for measured traces is left as future work (see subsection 5.4.3).

Per-Class Prediction Heatmaps. In addition to the averaged predicted class distributions, we also visualize the confusion behavior of the LDA model using heatmaps. Each heatmap shows the average



(a) Global LDA prediction distribution (simulated, std=3.0)



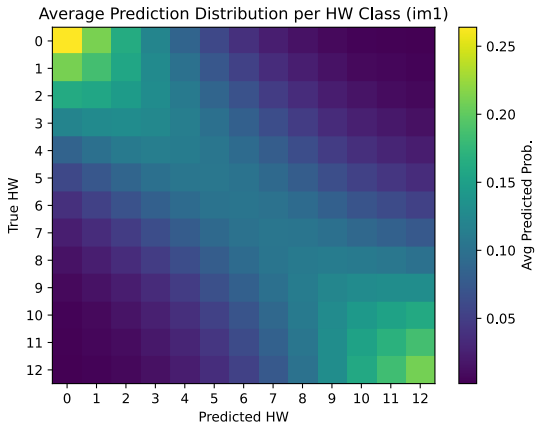
(b) Global LDA prediction distribution (simulated, std=1.0)

Figure 4.15 Global LDA prediction distributions over all POI sets and traces for `im1`. The model overestimates rare Hamming weights, even under low-noise simulation.

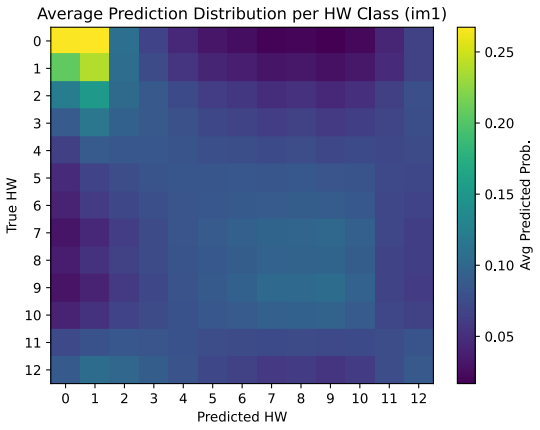
probability assigned to each predicted class (y-axis) for each actual Hamming weight class (x-axis). Figure 4.16 presents two representative examples: one for `im1` with simulated traces at $std = 3.0$ and one for `im2` from measured traces.

While for simulated traces with $std = 3.0$ a clear diagonal pattern emerges (indicating good distinguishability), the measured traces reveal fuzzier distributions with higher off-diagonal weight, confirming the degraded class separation in practice.

Further examples including `im2` for simulated traces and both `im1` and `im2` for measured and simulated conditions are included in section A.2.



(a) Heatmap for im1 (simulated traces, $std = 3.0$)



(b) Heatmap for im1 (measured traces, dsf=25)

Figure 4.16 Representative confusion heatmaps showing per-class prediction behavior of the LDA model. See section A.2 for full set.

4.5 Key Recovery Results

4.5.1 Attack Configurations and Results

We evaluate key recovery using various trace and parameter configurations. The attack uses profiling traces to train an LDA model and one attack trace to recover the full η vector. Recovery success is

verified by comparing the reconstructed secret with ground truth labels; in a real deployment, this could be checked via a signature verification step.

Due to the high computational effort and the large parameter space, we do not attempt exhaustive evaluations. Instead, we focused on combinations that, based on SNR and prediction evaluations, seemed most promising for 1-trace recovery using a minimal number of profiling traces. The test runs summarized in Table 4.7 reflect selected examples across varied settings as introduced in subsection 4.4.1. The table summarizes prediction scores and recovery outcomes of the tested configurations (all with a single attack trace).

Run	**Batch 1**		**Batch 2**		Result
	Rel. Diff (%)	Top/2nd (%)	Rel. Diff (%)	Top/2nd (%)	
1	0.18 / 1.36 / 0.72	8.30 / 8.24	9.57 / 11.6 / 10.57	12.39 / 11.08	✓
2	0.02 / 1.74 / 0.84	8.99 / 8.92	3.16 / 5.77 / 4.22	11.61 / 11.12	✓
3	0.36 / 1.32 / 0.87	8.92 / 8.84	1.29 / 2.00 / 1.66	8.60 / 8.45	✓
4	0.31 / 1.89 / 1.01	8.99 / 8.9	11.03 / 13.09 / 12.2	12.84 / 11.27	✓
5	0.00 / 2.00 / 0.92	6.97 / 6.90	0.16 / 3.00 / 1.02	6.79 / 6.72	✗
6	0.01 / 0.92 / 0.34	7.69 / 7.66	9.29 / 11.5 / 10.4	12.47 / 11.17	✗
7	0.00 / 0.36 / 0.11	7.88 / 7.88	0.00 / 0.95 / 0.20	10.35 / 10.33	✗

Table 4.7 Recovery success and prediction scores for one-trace attacks. "Rel. Diff" summarizes min / max / avg relative margins between 1st and 2nd candidates. "Top/2nd" lists average probabilities of top and second candidate. Note: The final column indicates whether all η_i values were correctly recovered.

We now highlight the factors influencing recovery performance.

Parameter selection. Our configuration choices were guided by observed SNR behavior across intermediates `im1` and `im2` (see Table 4.5). Downsampling factors (*dsf*) of 25, 50, and 125 were tested, with POI sets adapted accordingly to span approximately one clock cycle and capture peak SNR regions.

npoi, dsf, LDA_p. These parameters must be selected jointly, as they influence both spatial trace coverage. Although *dsf* 125 yielded high SNR values (see 4.3.3), it consistently underperformed not only for LDA predictions (4.4.1) but also at recovering the secret, with no 1-trace attack success in our tests. By contrast, lower-SNR configurations (e.g., *dsf* 10) sometimes achieved perfect recovery. This highlights that neither SNR nor class probability gaps alone predict recovery success reliably.

Stacking and POI selection. Vertical stacking was a critical enabler: none of the runs without stacking (e.g., test 5) succeeded, even with significantly more profiling data. Likewise, the rigid *fixed* POI selection strategy (in test 6) led to failure, suggesting that careful POI adaptation to leakage shape is important.

Attack traces. We observed that recovery was highly sensitive to the specific attack trace used. While some parameter settings succeeded with one trace, others required multiple. In many failing configurations, increasing the number of attack traces to two or more enabled recovery, even though the per-index predictions remained unchanged. Also individual differences of attack traces can play a role: several 1-trace attacks only work on a subset of the availabe attack traces. This suggests that certain individual traces have insufficient leakage or exhibit unfavorable misalignment.

Intermediates. In general, recovery of η_i values based on `im2` predictions was significantly more reliable than those from `im1`. This aligns with earlier observations about the clearer class separation and higher signal diversity in `im2`. The implications and causes of this difference are discussed throughout previous sections, in particluar 4.2.

Recovery vs prediction. Notably, high LDA prediction scores did not always correlate with successful η recovery. Some failed runs showed strong relative differences and high class confidence, while successful runs sometimes had smaller gaps. This disconnect may be due to a small number of incorrectly predicted indices that block full reconstruction, the non-uniform influence of each index, or trace-specific artifacts like noise or misalignment.

Moreover, we found that increasing the number of profiling traces could degrade LDA classification margins (possibly due to overfitting or added trace variability), yet still improve recovery outcomes. That better LDA predictions lead to worse outcomes of attacks is counterintuitive, since our candidates are directly based on accumulations of LDA predictions. This underlines a possibly complex relationship between profiling size, model quality, prediction margins, and actual key recovery performance.

Overall, these findings demonstrate that recovery success depends on a subtle interplay of many factors, including trace quality, model generalization, intermediate selection, and even attack trace randomness.

4.5.2 Threshold Setting

In theory, thresholding could improve the reliability of η_i predictions by rejecting low-confidence candidates. However, as already observed for simulated traces in subsection 4.1.2, the probability gap between the top and second-best hypotheses is typically too small to support practical thresholding (see Table 4.7).

While we can identify test configurations with thresholds of around 10% for the second η_i batch, the highest values we find for the first batch are areound 1% on average with a minimum of 0.31%.

This may be related to various observations: the SNR values for `im1` are consistently lower than for `im2`; the same holds true for LDA class predictions; the difficulty to differ between `im1` hypotheses likely contributes to these worse `im1` results. The low margins for the top candidates of η_i in the first batch reflect these previous obstacles.

Even when using 10 full attack traces, many η_i indices failed to exceed a confidence threshold of 20%. As a result, the final attack procedure uses simple maximum-likelihood selection per index, without rejection filtering.

5 Discussion

5.1 Implications of Findings

5.1.1 Feasibility of Side-Channel Attacks on CROSS

The unprotected reference implementation of CROSS clearly exposes side-channel leakage that can be exploited by attackers. As expected for cryptographic implementations without masking or hiding countermeasures, the scheme is vulnerable to profiling-based attacks. In our approach, the difficulty of distinguishing `im1` leakage values poses a notable challenge, yet this alone is not sufficient to prevent key recovery.

Our results confirm that CROSS implementations, in their current form, do not resist standard side-channel analysis techniques and require additional protection mechanisms to ensure implementation-level security.

We demonstrated the feasibility of template attacks, which rely on profiling traces to build a leakage model and require significantly fewer traces in the attack phase than DPA-style attacks.

A recent attack proposed in [12] also targets the same leakage source using a different strategy. That approach:

- uses Differential Power Analysis (DPA) and does not require a profiling device,
- operates on a single trace from the device under attack, and
- requires substantial computational effort for round alignment and hypothesis testing.

In contrast, our template-based attack requires roughly 5,000 profiling traces but is computationally efficient during the actual key recovery step. This trade-off makes it a realistic threat model in settings where device access for profiling is available.

5.1.2 Increasing Number of Attack Traces

We expected that the distinguishability for `im1` values would improve as the number of attack traces increased, since more traces create more opportunities for diverging hypotheses to produce different Hamming weights. However, this effect did not manifest in practice.

In one tested configuration using 5,000 profiling traces, both a 1-trace (test run 1 inTable 4.6) and an 8-trace attack resulted in full key recovery. Yet the prediction margins, particularly for batch 1 (`im1`), remained almost unchanged, and in some cases even degraded when using more attack traces.

This observation suggests that increasing the number of attack traces yields limited benefits in terms of distinguishability, at least for `im1`. Possible causes include:

- inherent limitations in signal separability due to `im1`'s low SNR,
- trace misalignment across vertically stacked samples,
- or structural issues in the hypotheses that prevent meaningful Hamming weight variation.

These findings align with earlier results on simulated traces (see subsection 4.1.2) and underline that improvements in recovery from more attack traces are not guaranteed. They also raise further questions about whether prediction scores can fully capture the distinguishability characteristics of the hypotheses involved.

5.1.3 Bottleneck

The primary bottleneck in our attack lies in recovering the first $\mathbf{u}_i$ batch, which is based on intermediate values `im1`. Despite `im2` being algebraically more complex, the initial batch is far more difficult to recover, and success depends critically on it.

This bottleneck stems from several combined factors:

- **Lower SNR and poorer leakage.** `im1` exhibits significantly lower SNR values than `im2`, as shown in 4.3.3. This results in weaker leakage and degraded LDA prediction quality for the first batch.
- **Fewer hypotheses to distinguish.** The first batch of $\mathbf{u}_i$ involves $(k - n)$ multiplications, while the second batch uses k values. This results in fewer Hamming weight variants per trace for `im1`, reducing the diversity of leakage patterns and making statistical separation harder.
- **Lower intrinsic distinguishability.** Multiplication without reduction tends to preserve Hamming weight classes, leading to many hypotheses for `im1` that produce nearly identical leakage. This property is not present in `im2`, where modular reduction introduces more variation and better separation between candidates.
- **Sequential dependency.** Recovery of $\mathbf{u}_i$ from `im1` must precede the processing of `im2`, since the latter depends on correct hypotheses from the former. An incorrect guess in the first batch propagates errors downstream and often leads to full attack failure.

We also verified that the poor leakage of `im1` is not caused by compiler optimizations or reordering. Even with disabled optimization (`-O0`), the leakage remained weak. Furthermore, while SNR is typically a good indicator of side-channel quality, in this case, it does not correlate well with distinguishability: SNR measures Hamming weight class separation for a fixed POI and Hamming weight value, but many `im1` hypotheses share the same Hamming weight, offering no separation regardless of SNR.

Overall, these factors make the first batch the dominant obstacle for reliable key recovery, and explain the limitations of thresholding and classifier confidence filtering discussed in subsection 4.5.2.

5.2 Potential Countermeasures

Our results confirm that CROSS, in its current form, requires implementation-level countermeasures to protect against side-channel attacks.

Reducing Leakage in CROSS-R-SDP. One concrete idea applicable to CROSS-R-SDP is to reduce the leakage in the second batch of the vector-matrix multiplication. In the reference implementation, elements $\mathbf{u}[\mathbf{K} : \mathbf{N} - \mathbf{1}]$ are used multiple times during the computation of $\mathbf{v} = \mathbf{u} \cdot V^{\mathsf{T}}$. This exposes them to repeated leakage.

The modified implementation in Listing 5.1 buffers all results in a temporary array and uses each $\mathbf{u}[\mathbf{K} : \mathbf{N} - \mathbf{1}]$ value only once, during the final addition. While this increases memory usage, it may help reduce exploitable leakage in the second batch of the computation.

```
1  void fq_vec_by_fq_matrix(uint8_t res[N-K], uint8_t u[N], uint8_t V_tr[K][N-K]) {
2    uint8_t temp[N-K] = {0};
3    for (int i = 0; i < K; i++) {
4      for (int j = 0; j < N-K; j++) {
5        temp[j] += u[i] * V_tr[i][j];  // mod 127
6      }
7    }
8    for (int j = 0; j < N-K; j++) {
9      res[j] = temp[j] + u[K+j];  // mod 127
10   }
11 }
```

Listing 5.1 Vector-matrix multiplication with reduced leakage

This countermeasure is specific to the CROSS-R-SDP variant, where the error vector comprises n values, k in batch 1 and $n - k$ in batch2. In contrast, the CROSS-RSDP(G) variants use error vectors that effectively comprise $m < k$ values. As a result, countermeasures aimed at the second batch, are ineffective for this variant.

Standard Countermeasures. Beyond this specific case, standard countermeasures such as masking, hiding, and operand shuffling may be introduced to protect implementations in practice. For example, randomizing the execution order of independent operations in the matrix multiplication may help prevent vertical or horizontal alignment.

We leave the integration and performance analysis of such countermeasures for future work.

5.3 Limitations of Current Work

5.3.1 Limited Exploration of Attack Surface

Our attack focuses on a specific leakage point: the vector-matrix multiplication used to compute $s = \mathbf{u} \cdot H^\top$. We did not evaluate alternative leakage vectors in practice, although we briefly outline such possibilities in subsection 5.4.2.

Additionally, we restricted our attack to rounds where $b = 1$. Operations with $b = 0$ (including matrix-vector products or other steps in the signature generation) were not investigated in detail, but we provide initial thoughts in subsection 5.4.1.

5.3.2 Exclusion of Alternative CROSS Configurations

In this thesis, we focused exclusively on attacking the CROSS-R-SDP-1-fast configuration. Other configurations were not investigated due to time constraints: trace collection, data management, and attack evaluation are resource-intensive. Nonetheless, the feasibility of adapting our attack to other CROSS configurations is discussed here and further addressed in section 5.4.

The following paragraphs outline what adapting the attack would entail. In simple terms, one would need to: compile a different CROSS configuration (with our attack-enabling modifications) as a shared library for the Python script; build and deploy the same configuration to the target microcontroller (STM32F4); and then capture profiling and attack traces with the new binary. While the attack script already adapts to different parameter sets via the shared library, some adjustments would still be necessary.

CROSS-R-SDP. Other configurations of CROSS-R-SDP can be targeted with the same attack technique. However, their effectiveness will vary depending on the parameter sets defined in Table 1.1. In particular, Figure 5.1 suggests that the *balanced* and *small* variants of CROSS-R-SDP-1 may leak more information due to a higher number of rounds and a larger share of those rounds using $b = 1$, without increasing the vector lengths of $\mathbf{u}$ and η.

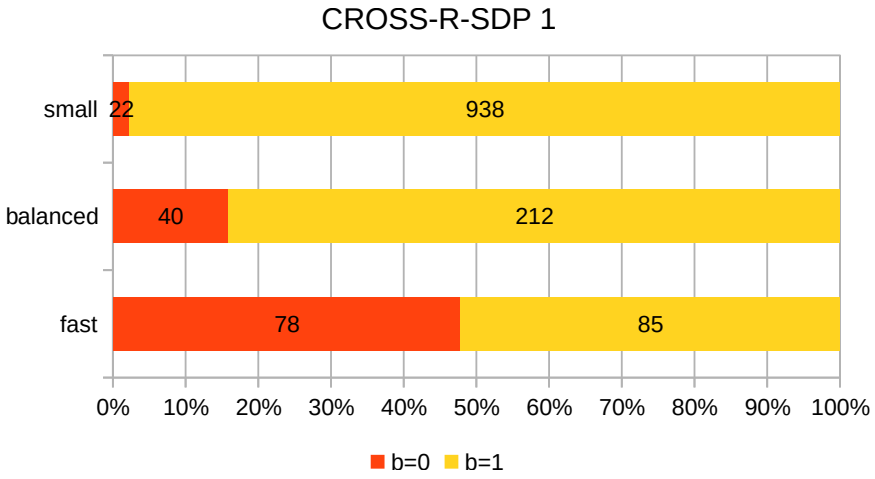


Figure 5.1 Weights for CROSS-R-SDP-1

Table 5.1 summarizes leakage-relevant characteristics for all CROSS-R-SDP configurations. The key metric shown is the number of intermediate values (e.g., `im1`, `im2`) that contribute to distinguishing each η_i. These numbers are proportional to $(n-k) \cdot w$ in each batch, and indicate how much usable leakage each η_i receives. The hypothesis space (determined by z and p) is constant across all CROSS-R-SDP variants, so the difficulty of distinguishing hypotheses from im1 leakage persists.

		Batch 1			**Batch 2**		
Security Category	**Optim. Corner**	**# u[i]**	**# im1 / u[i]**	**# im1 / eta[i]**	**# u[i]**	**# im2 / u[i]**	**# im2 / eta[i]**
		k	$n-k$	$(n-k) \cdot w$	k	$n-k$	$(n-k) \cdot w$
1	fast	76	51	4335	51	76	6460
	balanced	76	51	10812	51	76	16112
	small	76	51	47838	51	76	71288
3	fast	111	76	9652	76	111	14097
	balanced	111	76	25840	76	111	37740
	small	111	76	68932	76	111	100677
5	fast	150	101	17069	101	150	25350
	balanced	150	101	43127	101	150	64050
	small	150	101	92112	101	150	136800

Table 5.1 Leakage-relevant values for CROSS-R-SDP configurations ($p = 127, z = 7$).

The *balanced* and *small* variants of CROSS-R-SDP-1 offer more leakage without increasing the vector lengths of $\mathbf{u}$ and η. This should translate into improved recovery performance. For CROSS-R-SDP with security levels 3 and 5, the number of leakage values per hypothesis is also higher, but so is the number of η_i values to recover. In particular, the increased size of the first batch (k) may cancel out some of the gain in total leakage.

CROSS-R-SDP(G). Although CROSS-R-SDP(G) shares structural similarities with CROSS-R-SDP, it differs in several ways—most notably in its arithmetic operations and in the use of a larger field ($p = 509$) and hypothesis space ($z = 127$).

We did not perform a full analysis of CROSS-R-SDP(G) due to time limitations. However, the evaluation shown in Figure 5.2 suggests that distinguishability of `im1` hypotheses may improve compared to CROSS-R-SDP. This is due to a more diverse distribution of observed Hamming weights, which helps separate otherwise ambiguous hypotheses. Still, the hypothesis space is much larger and could mitigate those benefits.

5.4 Directions for Future Research

5.4.1 Extending the Current Attack to All Rounds

While this thesis focuses on rounds with $b = 1$, where the *round seed* is exposed in the signature, rounds with $b = 0$ can also be targeted using a similar approach. In these rounds, the challenges β and the vectors $\mathbf{y}$ are known, and σ is included in the signature. By hypothesizing candidate values for $\mathbf{u}$, one

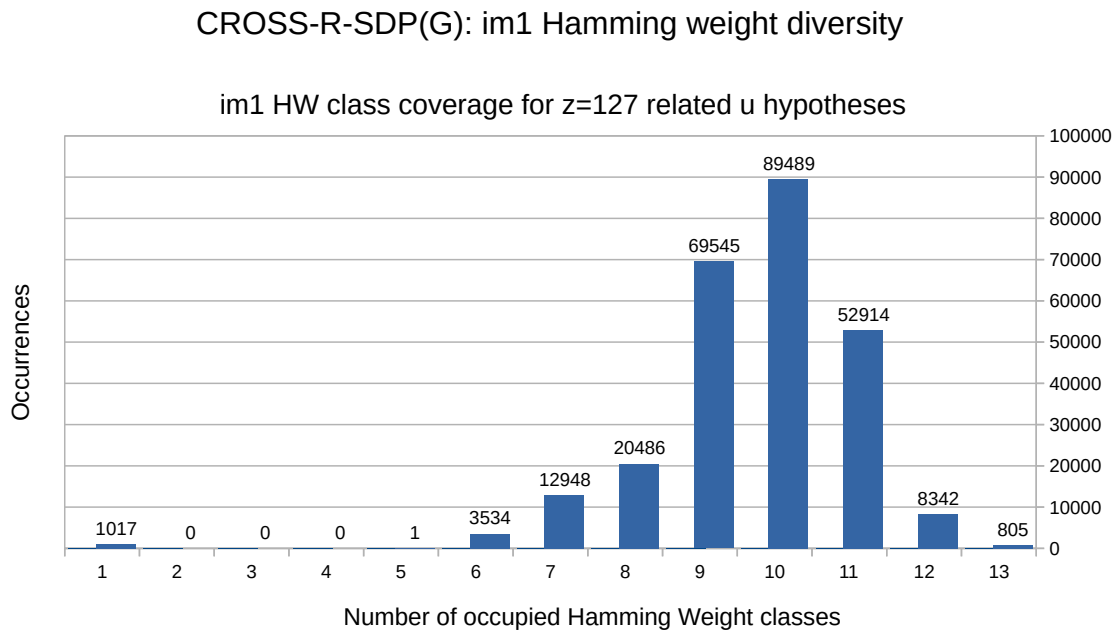


Figure 5.2 im1 Hamming weight coverage (CROSS-R-SDP(G))

can reuse the current LDA models to target the same intermediate values `im1` and `im2` . If a hypothesis is correct, it is possible to recover η' as follows:

$$\begin{aligned} \text{(known from signature)} \quad & \sigma,\ \mathbf{y},\ \beta \in \mathbb{F}_p^n \\ \text{(hypothesis)} \quad & \mathbf{u} \in \mathbb{F}_p^n \\ \text{(derived)} \quad & \mathbf{u}' = \sigma \star \mathbf{u} \mod p \\ & \eta' = (\mathbf{y} - \mathbf{u}')/\beta \mod p \\ & \eta = \sigma \star \eta' \mod z \end{aligned}$$

The recovery steps for $b = 0$ rounds involve hypothesizing $\mathbf{u}$ and using the known values σ and $\mathbf{y}$ from the signature. For each hypothesis $\mathbf{u}$, we compute the isometric form $\mathbf{u}' = \sigma \star \mathbf{u}$, followed by $\eta' = (\mathbf{y} - \mathbf{u}')/\beta$, and finally $\eta = \sigma \star \eta'$.

This process fails only when $\mathbf{u}_i = 0$, resulting in $\mathbf{v}_i = 0$ and making $\sigma_i = \mathbf{u}_i/\mathbf{u}'_i$ undefined—a rare case occurring in just $1/p$ of hypotheses. Unlike the $b = 1$ case, where σ must be recovered via logarithmic inversion of $\mathbf{v}$, the $b = 0$ case allows direct use of the known σ from the signature, simplifying the recovery process. The statistical indistinguishability between Hamming weight classes of intermediate values remains an obstacle, and fewer $b = 0$ rounds are typically available in a given signature.

Despite being less frequent (see Figure 5.1), rounds with challenge $b = 0$ can still contribute valuable leakage and increase the overall number of usable traces.

5.4.2 Additional Attack Surface Exploration

An advantage of the attack performed in this thesis is that the vector-matrix multiplication offers many leakage values that can be targeted. It comes with the drawback of difficulties to distinguish `im1` intermediates based on different hypotheses of `u` due to their often equal Hamming weights. The methods described in this section have different trade-offs, and it may be interesting to explore them in more detail to understand whether they provide a better attack surface.

To thoroughly investigate the vulnerabilities of CROSS for SCA in order to develop protection mechanisms, more research has to be done. Being among the first to look into this algorithm, we picked the seemingly easiest path to expose basic attack surface. That also means that we did not try all possibe attack paths.

The attack on values of $\mathbf{u}$ via `im1` has the disadvantage that, in most cases, the derived hypotheses `im1[i][j] = u[i] * V_tr[i][j]` have the same Hamming weights as the z left hypotheses of `u[i]`. This causes issues when using a Hamming weights leakage model and trying to distinguish the hypotheses to find a best candidate—the probability sum for hypotheses that share the same Hamming weight are equal. We can review the CROSS-ID algorithm to find values that are better suited for SCA; ideally, we require values that match the following properties:

- we can use them to derive the secret η
- we can build hypotheses on small vector elements and their results, independent of other values
- we get many leakage values; this is the case for values processed in iterative rounds of CROSS-ID
- the values are processed in operations, leading to derived hypotheses; either with other known values, or no other unknown values involved
- ideally, the derived hypotheses differ in their Hamming weights for original hypotheses with same Hamming weight

Table 5.2 lists the operations performed within rounds of CROSS-ID as specified in Algorithm 1. To find suitable values, we look at mathematical operations as follows:

- input values for those operations (to build an original hypothesis on them)
- other input values (can be either none or known values)
- the resulting value to build a derived hypothesis

Ref.	Operation	Leakage Potential
op1	`Sample Seed`$^{(i)} \xleftarrow{\$} \{0,1\}^{\lambda}$	✗ Unprocessed
op2	`Sample` $(\mathbf{e}'^{(i)}, \mathbf{u}'^{(i)}) \xleftarrow{\texttt{Seed}^{(i)}} G \times \mathbb{F}_p^n$	✗ Not directly used
op3	`Compute` $\sigma^{(i)}$ such that $\sigma^{(i)}(\mathbf{e}'^{(i)}) = \mathbf{e}$	✓ Targeted
op4	`Set` $\mathbf{u}^{(i)} = \sigma^{(i)}(\mathbf{u}'^{(i)})$	✓ Similar to above
op5	`Compute` $\tilde{\mathbf{s}}^{(i)} = \mathbf{u}^{(i)}\mathbf{H}^{\top}$	✓ Used in current approach
op6	$\mathtt{c}_0^{(i)} = \mathsf{Hash}(\tilde{\mathbf{s}}^{(i)}, \sigma^{(i)}, \mathtt{Salt}, i)$	✗ Not useful
op7	$\mathtt{c}_1^{(i)} = \mathsf{Hash}(\mathbf{u}'^{(i)}, \mathbf{e}'^{(i)}, \mathtt{Salt}, i)$	✗ Not useful

Table 5.2 Leakage usability of different operations in the CROSS signature generation process.

We look at the operations described in Table 5.2 and immediately can exclude a few for SCA:

- op1: this operation does not have an input value to build a hypothesis on, which increases the difficulty for attacking only hypothesized result values without correlated original hypotheses
- op2: *Seed* is the input value, and η' and $\mathbf{u}'$ are outputs; in rounds with $b = 1$, the *Seed* is made available via signature, so a successful attack on this value would provide additional knowledge only in rounds with $b = 0$ which
- op6: several unknown input values for hash function, and the compressed output does not allow attacks on separate values of the vectors (no divide-and-conquer)
- op7: poses the same challenges as the previous hash operation

For the following analysis of the operations we still consider, we transition from the high-level concepts to source code denominators:

Evaluation of op3. `Compute` $\mathbf{v}^{(i)} = \eta^{(i)} - \eta'^{(i)} \mod 7$

- known inputs: η' for $b = 1$, none for $b = 0$

- original hypotheses: build on $\eta[\mathbf{i}]$ for $b = 1$, and on both $\eta[\mathbf{i}]$ and $\tilde{\eta}[\mathbf{i}]$ for $b = 0$
- derived hypotheses (leakage values): $\mathbf{v}$

For rounds with challenge $b = 1$, we learn η' from the signature. We can hypothesize single elements of η independently and verify the hypotheses by exploiting leakage of $\mathbf{v}$ values. Successfully targeting $\mathbf{v}$ would allow the recovery of η via $\eta = \mathbf{v} + \eta' \mod 7$. If we focus on challenges $b = 1$, we can exploit w leakage values for each hypothesis of η per signing operation. For each value of η, we have z possible hypotheses. The targeted leakage values of $\mathbf{v}$ take only z possible values in $\mathbb{Z}_z$. In case of CROSS-R-SDP ($z = 7$) that results in only three different Hamming weights: 0, 1, and 2.

Both η and η' are uniformly distributed. Accordingly, the resulting $\mathbf{v} = \eta - \eta' \mod 7$ varies in Hamming weight across most hypotheses. As a result, HW-based distinguishability for op3 is better preserved. We are less likely to encounter clustered Hamming weights across hypotheses, and thus, side-channel evaluation of $\mathbf{v}$ becomes more informative.

For rounds with $b = 0$, the hypothesis space increases to $z * z$, because we have to pose hypotheses for both $\eta[\mathbf{i}]$ and $\tilde{\eta}[\mathbf{i}]$. This leads to many more possible hypotheses mapped to the same number of Hamming weight classes and is much more challenging.

Evaluation of op4. `Compute` $\mathbf{u}^{(i)} = \mathbf{v}^{(i)} \cdot \mathbf{u}'^{(i)} \mod 127$[1]

- known inputs: $\mathbf{u}'$ for $b = 1$, $\mathbf{v}$ for $b = 0$
- original hypotheses: $b = 1$: hypothesize $\mathbf{v}[\mathbf{i}]$; $b = 0$: hypothesize $\mathbf{u}'[i]$
- derived hypotheses (leakage values): $\mathbf{u}[\mathbf{i}]$

This approach allows attacking rounds with both $b = 1$ and $b = 0$ and hence provides, in one signing operation, t leakage values for each hypothesized value. Hypotheses on $\mathbf{v}[\mathbf{i}]$ and $\mathbf{u}'[i]$ can be tested independently, allowing for divide-and-conquer style attacks. The recovery path for η is the same as used in our current approach (explained in subsection 3.3.1). Each signing operation reveals t potential leakage points, one per value of $\mathbf{u}[i]$.

In rounds with $b = 1$, we hypothesize $\mathbf{v}[\mathbf{i}] \in \{1, 2, 4, 8, 16, 32, 64\}$ and compute corresponding $\mathbf{u}[i] = \mathbf{v}[\mathbf{i}] \cdot \mathbf{u}'[i] \mod 127$. Although this results in different $\mathbf{u}[i]$ values for each hypothesis, a thorough analysis shows that all such values have identical Hamming weights. This invariance holds for all tested values of $\mathbf{u}'[i]$ and stems from the structure of the multiplication: all $\mathbf{v}[\mathbf{i}]$ values are powers of two, and the reduction logic applied (`FQRED_DOUBLE`) preserves the population count in all occurring cases.

As a result, the derived hypotheses collapse into a single Hamming weight class, completely defeating HW-based distinguishability in op4 for rounds with $b = 1$. This behavior is even more restrictive than in the current approach based on `im1`, where some variation in Hamming weight was observed.

In rounds with $b = 0$, the attacker hypothesizes over all 127 possible values of $\mathbf{u}'[i]$ for a fixed, known $\mathbf{v}[\mathbf{i}]$. However, the resulting $\mathbf{u}[i] = \mathbf{v}[\mathbf{i}] \cdot \mathbf{u}'[i]$ still exhibits the same invariance: values that differ in $\mathbf{u}'[i]$ frequently yield outputs with identical Hamming weights. As with $b = 1$, this is due to the structure-preserving nature of multiplication by powers of two and the low-disruptive reduction logic.

Thus, targeting $\mathbf{u}$ directly appears to be a less promising path for side-channel analysis in this configuration.

5.4.3 LDA Model Verification

Our LDA model showed plausible behavior under low-noise simulated conditions. However, for captured traces, prediction accuracy degraded noticeably due to noise and misalignment. To assess the trustworthiness of LDA predictions, we compared the output distributions from simulated traces under high noise levels with those from measured traces. Interestingly, the predictions for measured traces often resembled those of simulated traces at medium to high noise levels (e.g., $\sigma = 3.0$). This suggests

[1] Not a conventional modulo operation. In the CROSS reference implementation, values ≥ 127 are reduced using a custom double-fold reduction called `FQRED_DOUBLE`, which preserves population count (Hamming weight) more than standard modular arithmetic.

that the degradation in prediction quality for real traces stems from noise and other distortions rather than modeling flaws.

A potential strategy for improving trace comparability, and indirectly verifying LDA behavior, would be to reduce measurement noise through trace averaging. Prior work [9] has demonstrated that averaging repeated measurements with fixed inputs can significantly reduce effective noise, thereby yielding sharper and more reliable classification outputs. For example, noise on an STM32F4 microcontroller was reduced from $\sigma = 0.4 \ldots 3.0$ to $\sigma = 0.2 \ldots 1.3$ by averaging ten such traces.

5.4.4 Bayesian Postprocessing of Predictions

A potential improvement for prediction and candidate selection is to apply Bayes' rule as a postprocessing step to LDA prediction probabilities.

Currently, we rely solely on the class probabilities output by the LDA model. However, some intermediate values (in particular, their Hamming weights) are statistically rare—they occur with very low frequency over all possible combinations (see subsection 3.5.2). We have observed cases where the LDA model assigns high probability to such rare classes, even though they appear extremely infrequently under the true distribution.

This suggests a possible mismatch between the model's prediction confidence and the global likelihood of observing a given class. In particular, falsely selecting a rare class based on model output can disproportionately affect recovery, especially for indices with ambiguous leakage.

To address this, one could adjust the LDA probabilities using prior knowledge about class frequency. Formally, Bayes' rule would allow computing:

$$P(\eta_i = c \mid \text{trace}) \propto P(\text{trace} \mid \eta_i = c) \cdot P(\eta_i = c)$$

where $P(\eta_i = c)$ reflects the empirical frequency of class c, derived from the distribution of Hamming weights across all hypotheses.

This approach penalizes classes that are rarely correct in practice, even if the model predicts them confidently. The adjustment could reduce the selection of unlikely values that otherwise dominate due to noise or overfitting.

However, caution is needed: if a truly rare class is correct, applying strong priors might suppress it, leading to systematic rejection of correct candidates. Evaluating this technique would require empirical testing, ideally comparing standard LDA-based predictions with Bayes-adjusted scores using labeled traces.

6 Conclusion

6.1 Summary of Contributions

This thesis demonstrates a practical profiling side-channel attack against the unprotected reference implementation of CROSS-R-SDP. We focused on a template-based approach using linear discriminant analysis (LDA), targeting leakage during vector-matrix multiplication.

We developed an efficient attack pipeline that:

- uses a small number of profiling and attack traces,
- exploits structured leakage from intermediate values in the CROSS signature scheme, and
- successfully recovers the secret vector η, even when per-index prediction confidence is low and no thresholding is applied.

We evaluated key parameters such as POI selection, downsampling factors, and explored batching strategies and efficient use of profiling traces.

Our analysis revealed that the first batch of intermediate values (`im1`) constitutes the main bottleneck due to limited leakage and poor separability. The second batch (`im2`) consistently allowed high-confidence predictions.

We also explored recovery performance under real-trace conditions and confirmed the feasibility of single-trace attacks with sufficient profiling, despite irregular relationships between LDA prediction quality and recovery success.

We looked at other attack vectors (subsection 5.4.2) and the bottleneck (subsection 5.1.3). Most likely our attack scenario is the most promising of the ones we have shortly explored.

6.2 Significance of Research

This work shows that the CROSS signature scheme, although secure in theory, is vulnerable to concrete side-channel attacks if implemented without countermeasures.

Our attack scenario—profiling with access to a reference device—is realistic in many embedded environments and highlights the need for implementation-level protections also in post-quantum cryptographic schemes.

The analysis also reveals new challenges in model-based side-channel attacks, particularly the difficulty of linking classifier outputs to recovery success when leakage is weak or ambiguous.

Beyond practical attack results, our study offers a structured methodology for side-channel analysis, and outlines paths for further exploration and countermeasures for CROSS.

6.3 Closing Remarks

Side-channel resistance remains a critical consideration in the design and deployment of cryptographic implementations. As post-quantum schemes transition toward standardization and real-world use, their resistance to physical attacks must be evaluated alongside mathematical security.

This work contributes to that effort by demonstrating both the practicality and limits of template-based attacks against CROSS. Future research may refine model robustness, improve leakage mitigation, or extend this methodology to other code-based schemes.

Ultimately, protecting post-quantum cryptography requires attention not only to algorithms, but also to how they leak.

List of Figures

List of Tables

List of Algorithms

Listings

Acronyms

CMOS complementary metal-oxide-semiconductor

CROSS the Codes and Restricted Objects Signature Scheme

CSPRNG cryptographically secure pseudorandom number generator

DPA differential power analysis

DSO digital storage oscilloscope

DUT device under test

ECC elliptic curve cryptography

EM electromagnetic

EMA electromagnetic analysis

HDF5 Hierarchical Data Format 5

HW Hamming weight

KEM key encapsulation mechanism

LDA linear discriminant analysis

MCU microcontroller unit

NIST the National Institute of Standards and Technology

POI point of interest

PQC post-quantum cryptography

R-SDP the restricted syndrome decoding problem

R-SDP(G) the R-SDP in a Subgroup

SCA side-channel attack

SCALib the side-channel analysis library

SDP syndrome decoding problem

SNR signal-to-noise ratio

SPA simple power analysis

TRNG true random number generator

TUM the Technical University of Munich

ZKP zero-knowledge proof

References

[1] Marco Baldi et al. *CROSS – Algorithm Specifications and Supporting Documentation*. en. 2023. https://www.cross-crypto.com/ (visited on 08/29/2024).

[2] Gaëtan Cassiers and Olivier Bronchain. "SCALib: A Side-Channel Analysis Library". en. In: *Journal of Open Source Software* 8.86 (June 2023).

[3] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. "Template Attacks". en. In: *Cryptographic Hardware and Embedded Systems - CHES 2002*. Ed. by Burton S. Kaliski, çetin K. Koç, and Christof Paar. Berlin, Heidelberg: Springer, 2002.

[4] Information Technology Laboratory Computer Security Division. *Call for Proposals - Post-Quantum Cryptography | CSRC | CSRC*. EN-US. 2016. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/Call-for-Proposals (visited on 04/19/2024).

[5] Information Technology Laboratory Computer Security Division. *Call for Proposals - Post-Quantum Cryptography: Additional Digital Signature Schemes | CSRC | CSRC*. EN-US. Aug. 2022. https://csrc.nist.gov/Projects/pqc-dig-sig/standardization/call-for-proposals (visited on 02/18/2025).

[6] Amos Fiat and Adi Shamir. "How To Prove Yourself: Practical Solutions to Identification and Signature Problems". en. In: *Advances in Cryptology — CRYPTO' 86*. Ed. by Andrew M. Odlyzko. Berlin, Heidelberg: Springer, 1987.

[7] Pierre-Alain Fouque et al. "Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU". In: 2019.

[8] Lov K. Grover. "A fast quantum mechanical algorithm for database search". In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing*. STOC '96. New York, NY, USA: Association for Computing Machinery, July 1996.

[9] Mike Hamburg et al. "Chosen Ciphertext k-Trace Attacks on Masked CCA2 Secure Kyber". en. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (Aug. 2021).

[10] Paul Kocher, Joshua Jaffe, and Benjamin Jun. "Differential Power Analysis". en. In: *Advances in Cryptology — CRYPTO' 99*. Ed. by Michael Wiener. Berlin, Heidelberg: Springer, 1999.

[11] Paul C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". en. In: *Advances in Cryptology — CRYPTO '96*. Ed. by Neal Koblitz. Berlin, Heidelberg: Springer, 1996.

[12] Jonas Schupp and Georg Sigl. *A Horizontal Attack on the Codes and Restricted Objects Signature Scheme (CROSS)*. Publication info: Preprint. 2025. https://eprint.iacr.org/2025/116 (visited on 05/05/2025).

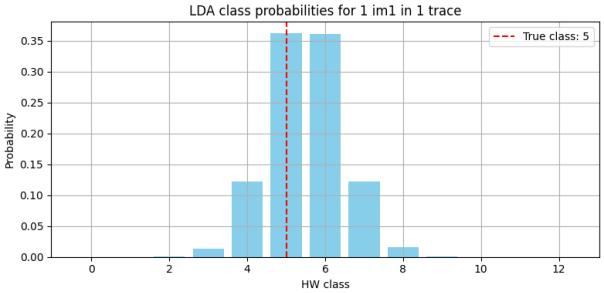
[13] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Review* 41.2 (Jan. 1999). Publisher: Society for Industrial and Applied Mathematics.

[14] National Institute of Standards and Technology. *Module-Lattice-Based Digital Signature Standard*. en. Tech. rep. Federal Information Processing Standard (FIPS) 204. U.S. Department of Commerce, Aug. 2024. https://csrc.nist.gov/pubs/fips/204/final (visited on 05/06/2025).

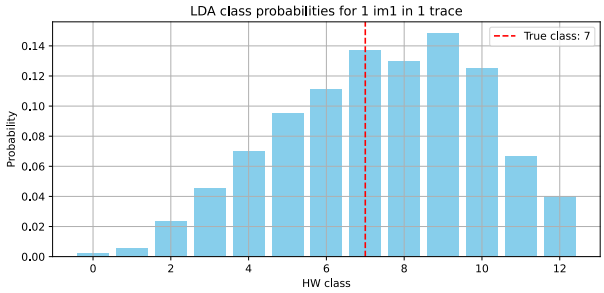
[15] National Institute of Standards and Technology. *Stateless Hash-Based Digital Signature Standard*. en. Tech. rep. Federal Information Processing Standard (FIPS) 205. U.S. Department of Commerce, Aug. 2024. https://csrc.nist.gov/pubs/fips/205/final (visited on 05/06/2025).

A Additional LDA Prediction Plots

A.1 LDA Predictions for Individual Samples



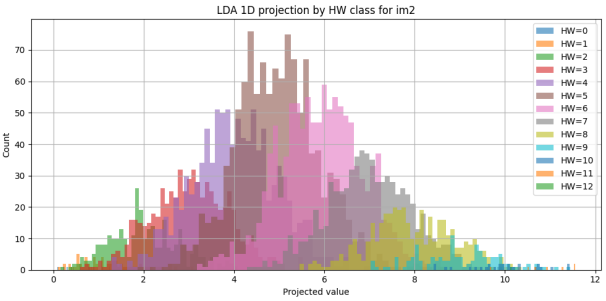
(a) Single simulated im1 sample (std=1.0): predicted probabilities for all Hamming weights.



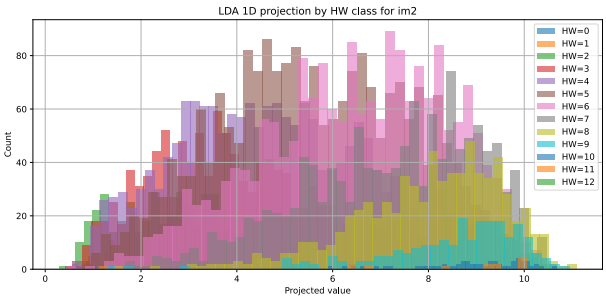
(b) Single captured im1 sample (dsf=25): predicted probabilities for all Hamming weights.

Figure A.1 LDA probability predictions for a single trace sample, showing likelihood for each Hamming weight (HW) class.

A.2 Statistical Analysis of Prediction Distributions

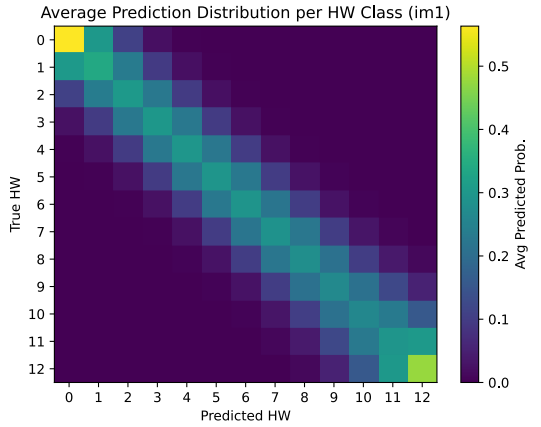


(a) Simulated im2 traces (std=1.0): average LDA predictions grouped by actual HW class.

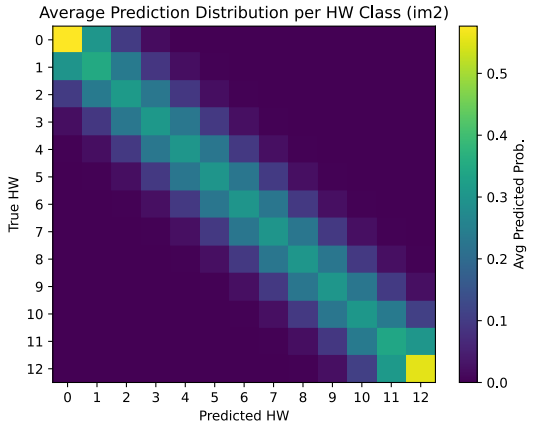


(b) Captured im2 traces (dsf=25): average LDA predictions grouped by actual HW class.

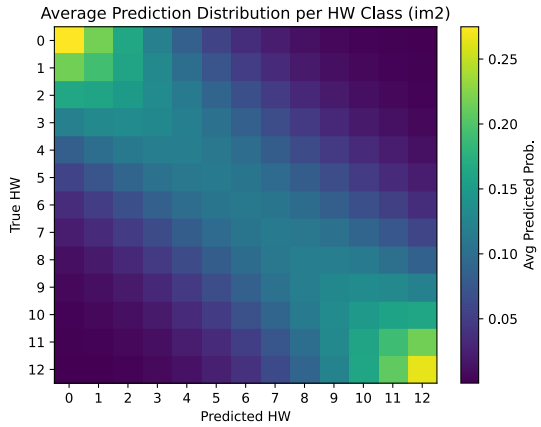
Figure A.2 Distributions of predicted probabilities per HW class: for each actual HW value, the average LDA-assigned probability for all classes is shown.



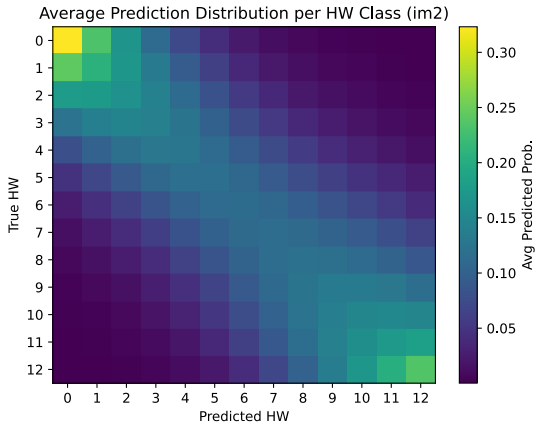
(a) Simulated trace (std=1.0), im1



(b) Simulated trace (std=1.0), im2

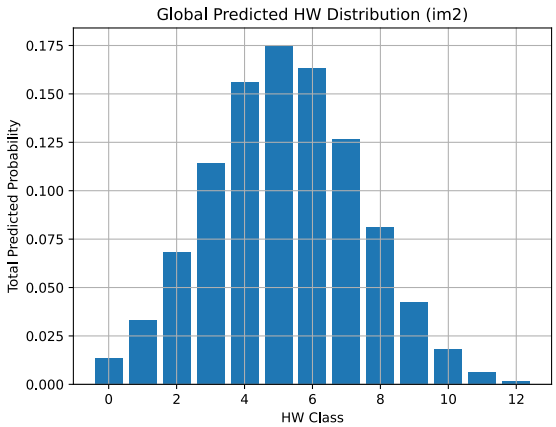


(c) Simulated trace (std=3.0), im2

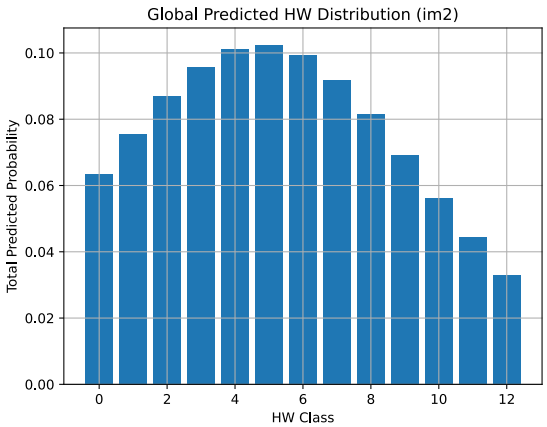


(d) Measured trace (dsf=25), im2

Figure A.3 Additional global heatmaps of predicted probabilities per true HW class, for various inputs and models.

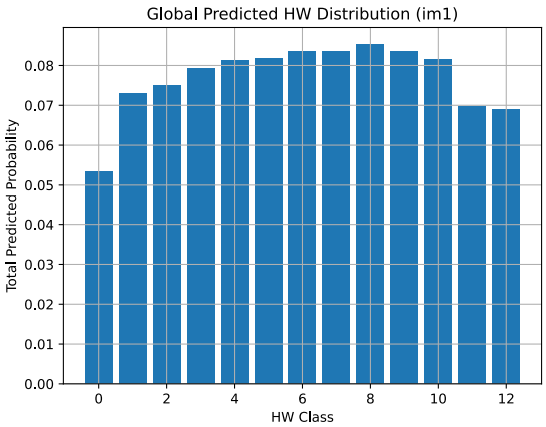


(a) Simulated traces (std=1.0): overall HW prediction distribution.

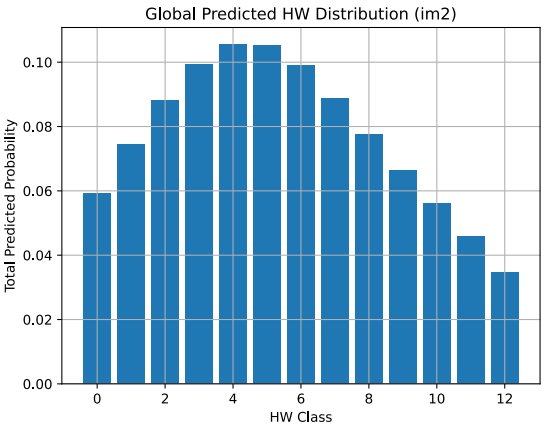


(b) Simulated traces (std=3.0): overall HW prediction distribution.

Figure A.4 Global prediction distribution (im2): sum of predicted probabilities for each HW class across all traces.



(a) Captured traces (dsf=25, npoi=7, im1): overall HW prediction distribution.



(b) Captured traces (dsf=25, npoi=7, im2): overall HW prediction distribution.

Figure A.5 Global prediction distribution: sum of predicted probabilities for each HW class across all traces.