Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Nora Mosand Viken

# Power-Based Side-Channel Evaluation of MAYO and Masking Countermeasures

Master's thesis in Cyber Security and Data Communication
Supervisor: Tjerand Silde
Co-supervisor: Andreas Sandø Krogen and Ella Moe Wolff
June 2026

NTNU
Norwegian University of Science and Technology

Nora Mosand Viken

# Power-Based Side-Channel Evaluation of MAYO and Masking Countermeasures

Master's thesis in Cyber Security and Data Communication
Supervisor: Tjerand Silde
Co-supervisor: Andreas Sandø Krogen and Ella Moe Wolff
June 2026

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

# Problem description

The ongoing development of quantum computers challenges the security assumptions underlying widely deployed public-key cryptographic systems. In response, post-quantum cryptography seeks to develop cryptographic algorithms that remain secure in the presence of quantum adversaries. As part of this transition, the National Institute of Standards and Technology (NIST) has played a central role in the standardization of post-quantum public-key algorithms.

To strengthen the resilience of post-quantum cryptographic standards, NIST has emphasized algorithmic diversity for Digital Signature Algorithms (DSAs) through an additional signature initiative. Although lattice-based approaches are widely trusted and form the basis of several standardized post-quantum algorithms, this effort seeks to reduce dependence on a single cryptographic family. As a result, several alternative families are being explored, including multivariate-based, MPC-in-the-head, code-based, and isogeny-based signature schemes.

Although the theoretical security of these schemes has been extensively studied, considerably less attention has been paid to their resistance against implementation-level attacks. Experience from classical cryptography has shown that physical implementations may leak information through side channels such as power consumption, or be vulnerable to active attacks such as fault injections, enabling adversaries to bypass the intended security guarantees of the scheme. For many post-quantum algorithms, these practical security aspects remain insufficiently explored.

This master's thesis focuses on the experimental evaluation of side-channel vulnerabilities in implementations of post-quantum DSAs from the multivariate family. By conducting side-channel measurements using a ChipWhisperer platform, the project aims to contribute to a broader understanding of the implementation-level security of multivariate signature schemes.

*Approved: 2026-02-17 – Tjerand Silde, NTNU (Main supervisor)*

## Abstract

The development of quantum computers motivates a transition in public-key cryptography, where widely used assumptions based on the computational hardness of integer factorization and discrete logarithms are being replaced by alternative hardness assumptions believed to resist quantum attacks. As part of this shift, several post-quantum digital signature candidates have been proposed, including schemes from the multivariate family. However, replacing the underlying mathematical problem is only one part of the transition. To support secure practical deployment, the evaluation of post-quantum implementations should also extend to physical attacks. In this thesis, we address this aspect for multivariate post-quantum signatures by experimentally evaluating power-based side-channel leakage in the MAYO digital signature scheme.

We reproduce the non-profiled single-trace Correlation Power Analysis (CPA) attack presented by Vishwaajith et al. (IACR ePrint 2026/228) on the nibble-sliced MAYO implementation using a ChipWhisperer-based measurement setup. The attack targets a secret-dependent operation in the MAYO secret key expansion, where public matrix data are multiplied with the secret oil matrix. By exploiting repeated executions of this operation within one execution of the target function, entries of the secret oil matrix can be recovered one coefficient at a time. We further evaluate masking as a countermeasure against the attack. Masked versions of the targeted operation are implemented using two and three shares, and the attack strategy is adapted to these settings.

The results show that the targeted MAYO implementation leaks exploitable information through power consumption. The reproduced CPA attack distinguishes correct oil coefficients from competing hypotheses and scales from recovery of individual coefficients to recovery of the full secret oil matrix. From the attacker's perspective, the main practical limitation is not the absence of leakage, but the difficulty of correctly segmenting noisy traces for further processing. The masking evaluation shows that masking increases the complexity of the attack, but does not prevent recovery in the tested implementation. For both two and three shares, the individual shares can be attacked separately and recombined to recover the secret oil values, provided that the adversary can correctly perform the required trace preprocessing. These results suggest that masking alone is not sufficient against the considered single-trace attack model. However, masking may still provide a useful foundation when combined with hiding techniques that disturb the preprocessing step.

## Sammendrag

Utviklingen av kvantedatamaskiner motiverer et skifte innen offentlig nøkkel-kryptografi, der mye brukte hardhetsantakelser basert på heltallsfaktorisering og diskrete logaritmer erstattes av alternative problemer som antas å være motstandsdyktige mot kvanteangrep. Dette har resultert i en rekke kandidater for kvantesikre digitale signaturer, inkludert algoritmer fra den multivariate familien. Å erstatte det underliggende matematiske problemet er imidlertid bare én del av overgangen. For at signaturalgoritmene skal forbli sikre i praksis, bør evalueringen av kvantesikre implementasjoner også omfatte fysiske angrep. I denne masteroppgaven undersøker vi dette aspektet for multivariate signaturer ved å eksperimentelt evaluere sidekanallekkasje i strømmålinger fra den digitale signaturalgoritmen MAYO.

Vi reproduserer angrepet presentert av Vishwaajith et al. (IACR ePrint 2026/228), et ikke-profilert sidekanalangrep basert på et enkelt strømspor mot MAYO-implementasjonen med nibble-sliced datarepresentasjon. Angrepet retter seg mot en hemmelighetsavhengig operasjon i nøkkelutvidelsen til MAYO, der offentlige matrisedata multipliseres med den hemmelige oljematrisen. Ved å utnytte gjentatte utførelser av denne operasjonen i én signering kan elementer i den hemmelige oljematrisen gjenopprettes en etter en. Videre evaluerer vi maskering som et mottiltak mot dette angrepet. Maskerte versjoner av den aktuelle operasjonen implementeres med to og tre maskeringsandeler, og angrepsstrategien tilpasses disse oppsettene.

Resultatene viser at strømmålingene fra den evaluerte implementasjonen av MAYO inneholder informasjon som kan utnyttes av en angriper. Det reproduserte angrepet skiller korrekte oljekoeffisienter fra konkurrerende hypoteser og skalerer fra gjenoppretting av individuelle koeffisienter til gjenoppretting av hele den hemmelige oljematrisen. Fra angriperens perspektiv er den viktigste praktiske begrensningen ikke fravær av lekkasje, men vanskeligheten med å segmentere støyfulle strømspor korrekt for videre steg i angrepet. Maskeringsevalueringen viser at maskering øker kompleksiteten i angrepet, men ikke forhindrer gjenoppretting i den testede implementasjonen. For både to og tre maskeringsandeler kan de individuelle andelene angripes separat og rekombineres for å gjenopprette de hemmelige oljeverdiene, forutsatt at angriperen kan utføre den nødvendige forbehandlingen av strømsporene korrekt. Resultatene tyder på at maskering alene ikke er tilstrekkelig mot den vurderte angrepsmodellen. Maskering kan likevel være nyttig i kombinasjon med teknikker som gjør strømsporene vanskeligere å segmentere og analysere.

# Preface

This thesis marks the end of five rewarding years of study at the Norwegian University of Science and Technology (NTNU) in Trondheim, within the Department of Information Security and Communication Technology. The work has been a valuable final part of my studies, with a steep learning curve and the opportunity to explore a topic that has been both challenging and engaging.

I am sincerely grateful to my supervisors, Tjerand, Andreas and Ella, for their patience, feedback and support throughout this work. I greatly appreciate the valuable input, discussions and knowledge they have shared, which have helped me navigate practical and theoretical challenges during this thesis. Special thanks to Tjerand for making the Crypto-Lab and additional resources available throughout the semester.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# List of Acronyms

**ADC** Analog-to-Digital Converter.

**CPA** Correlation Power Analysis.

**DPA** Differential Power Analysis.

**DSA** Digital Signature Algorithm.

**ECC** Elliptic Curve Cryptography.

**FPU** Floating-Point Unit.

**HAL** Hardware Abstraction Layer.

**HW** Hamming Weight.

**M4R** Method of the Four Russians.

**MPKC** Multivariate Public Key Cryptography.

**MQ** Multivariate Quadratic.

**NIST** National Institute of Standards and Technology.

**OV** Oil and Vinegar.

**PKC** Public Key Cryptography.

**PoI** Point of Interest.

**PQC** Post-Quantum Cryptography.

**RAM** Random Access Memory.

**RNG** Random Number Generator.

**RSA** Rivest-Shamir-Adleman.

**SAD** Sum of Absolute Differences.

**SCA** Side-Channel Attack.

**SDG** Sustainable Development Goal.

**UART** Universal Asynchronous Receiver-Transmitter.

**UOV** Unbalanced Oil and Vinegar.

# Chapter 1
# Introduction

Digital signatures help establish trust in digital systems. They allow a verifier to check the origin and integrity of data, such as software updates, messages, and digital transactions [PPG24, p. 299-304]. This trust relies on the secrecy of the signing key. If the key is recovered by an adversary, the security guarantee provided by the signature scheme no longer holds [Sma16, p. 217]. One may believe that resistance against such key recovery depends only on the mathematical strength of the underlying scheme. However, when an algorithm is incorporated into real-world devices, the threat model takes another form. Physical implementations may leak information through side effects such as variations in power consumption [OG21, p. 1]. An attacker with access to the power supply of a device performing a signing operation may be able to measure these variations. If they depend on secret intermediate values, the attacker may recover information about the signing key. This type of attack is known as a Side-Channel Attack (SCA) [OG21, p. 1].

The transition to Post-Quantum Cryptography (PQC) [Nat17] makes this concern increasingly relevant. Future quantum computers are expected to threaten widely used public-key cryptographic schemes, which means that new signature algorithms must be developed, standardized, and eventually deployed [Nat17; BBD09]. Replacing today's algorithms with post-quantum alternatives therefore requires more than mathematical security arguments. For the schemes to be trusted in practice, their implementations must also be evaluated against physical attacks. For established cryptographic algorithms, SCAs and countermeasures have been studied for many years, leading to implementation techniques that reduce or hide secret-dependent leakage [ZF05]. Post-quantum schemes, however, are newer [Nat24b] and have not yet undergone the same level of implementation security analysis. This makes side-channel analysis of post-quantum digital signatures an important step toward building trust in future cryptographic systems.

In this thesis, we study this question for multivariate digital signatures through power-based side-channel analysis. This family is interesting because several multivariate

candidates have been considered in NIST's additional digital signature standardization process [Nat26]. Studying one concrete candidate may therefore give insight into implementation security challenges that are relevant beyond a single algorithm. MAYO [BCC+25] is chosen as the target in this thesis because it is a concrete multivariate signature scheme with publicly available implementations suitable for experimental side-channel evaluation.

## 1.1 Quantum Computing and Its Impact on Cryptography

Quantum computing represents a fundamental shift in computation, and its potential to render modern cryptographic standards vulnerable is a critical and widely recognized consequence of this development [BBD09; Nat17]. Unlike classical computers, the basic unit for quantum computers is the qubit, which is the quantum counterpart to the classical bit [Nan20, p. 942]. Whereas a classical bit is always in one of two basis states, either 0 or 1, a qubit can exist in a superposition of these states [Nan20, p. 944]. Superposition means that the state of a qubit can be described as a linear combination of 0 and 1, allowing quantum systems to represent and process information in ways that are fundamentally different from classical computers [Nan20, p. 942]. This property is often associated with quantum parallelism, where quantum algorithms can explore certain computational possibilities more efficiently than classical algorithms [BBD09, p. 19].

The cryptographic relevance of quantum computing became particularly clear in 1994, when Peter Shor introduced a quantum algorithm capable of solving the integer factorization problem and the discrete logarithm problem in polynomial time [Sho97; BBD09]. These two mathematical problems form the security foundation of several widely deployed public-key cryptographic systems. However, Shor's algorithm requires a sufficiently large quantum computer, which has not yet been realized at the scale needed to break modern cryptographic systems. For many years, this threat was therefore regarded as a distant prospect, but the rapid advancement of quantum technology has transformed it into a pressing concern for the near future [Jaq25; Ive25]. For cryptography, this means that long-standing assumptions about what is computationally infeasible must be reconsidered.

Many security mechanisms used in modern digital systems rely on Public Key Cryptography (PKC). By using pairs of mathematically related public and private keys, PKC enables functionalities such as secure communication, authentication, and digital signatures. Among these, digital signatures are essential for ensuring authenticity and integrity of information [Sma16, p. 217]. For several decades, digital signature schemes based on Rivest-Shamir-Adleman (RSA) [RSA78] and Elliptic Curve Cryptography (ECC) [Nat23], have been dominant mechanisms for providing these security guarantees. The challenge, however, is that the same mathematical

assumptions that make these systems secure against classical computers are the ones that could be solved by Shor's algorithm on a fault-tolerant quantum computer, as RSA and ECC rely on the integer factorization and the discrete logarithm problem, respectively. This creates a direct link between advances in quantum computing and the need to replace current public-key cryptographic infrastructure.

The point at which a quantum computer becomes capable of breaking widely used public-key cryptographic schemes is often referred to as "Q-Day" [Ive25]. The exact timing of such an event remains difficult to predict, and existing estimates vary. Some predictions suggest that cryptographically relevant quantum computers may emerge within the coming decade [Ive25], while others consider the timeline more uncertain [Jaq25]. Nevertheless, there seems to be a consensus between researchers and security authorities that migration must begin well before such machines exist, as the logistics of changing every cryptosystem will take significant time.

A proactive response to this threat is the foundation of PQC. PQC focuses on developing and deploying cryptographic algorithms believed to be secure against both classical and quantum adversaries while preserving essential security mechanisms [BBD09]. Governmental and institutional initiatives further emphasize the critical relevance and urgency of this field. For instance, the European Commission [Eur25], the Norwegian National Security Authority [Nas25], and the United States [Nat17] have all issued strategic roadmaps and recommendations for quantum migration. These coordinated efforts illustrate the profound global impact of quantum computing and the increasing necessity of prioritizing the secure implementation and deployment of quantum-resistant algorithms.

## 1.2 NIST Standardization Process

The standardization of PQC is a central part of the global response to the threat that quantum computers pose to digital security. In 2016, the National Institute of Standards and Technology (NIST) took its first formal step toward addressing this threat by initiating a public process for post-quantum cryptographic standardization [Nat16]. The goal of this process was to identify and standardize quantum-resistant public-key algorithms that could replace vulnerable cryptographic mechanisms used in modern digital systems. Since then, NIST has standardized both key encapsulation mechanisms and digital signatures, providing an important basis for the transition toward PQC [Nat24b].

For digital signatures, NIST has standardized ML-DSA [Nat24a] and SLH-DSA [Nat24c], based on CRYSTALS-Dilithium and SPHINCS+, respectively, while Falcon remains under standardization as FN-DSA [Nat17]. These schemes provide important post-quantum alternatives to classical signature schemes [Nat24b]. At the same time,

the selection revealed a need for greater diversity in the post-quantum signature portfolio. Two of the selected signature schemes, based on CRYSTALS-Dilithium and Falcon, rely on structured lattice assumptions [Nat24b; Eur23]. While lattice-based cryptography has been prominent in the NIST process and is associated with efficient implementations and practical performance [Eur23], relying too strongly on one family of mathematical assumptions may reduce long-term resilience. If weaknesses were discovered in this family, standardized alternatives based on different assumptions would be important.

To strengthen this diversity, NIST launched a separate process for additional post-quantum digital signature schemes in 2022 [Nat22]. This process focuses on identifying further signature schemes that can complement the already selected standards. NIST has in particular expressed interest in general-purpose signatures based on other assumptions than lattices, as well as schemes with short signatures and efficient verification [Nat22; ABC+24].

This thesis builds on the preliminary project [Vik25] by focusing on the multivariate candidates that progressed to the second round of NIST's additional signatures process [ABC+24]. The recent announcement of the third round of this process, in May 2026, further motivates this focus. After the second round, NIST selected nine out of fourteen candidates to continue to the third round [Nat26]. Among these, four belong to the multivariate family, namely MAYO, QR-UOV, SNOVA, and UOV. Since all multivariate candidates from the second round were selected to continue, the third-round announcement indicates continued interest in multivariate schemes as candidates for post-quantum digital signatures.

## 1.3 Research Objective

The objective of this thesis is to investigate the practical side-channel security of multivariate post-quantum digital signature candidates in NIST's additional signatures process, using MAYO as the concrete target for experimental evaluation. The thesis focuses on power-based side-channel leakage and evaluates whether a ChipWhisperer-based measurement setup can capture exploitable leakage from an embedded implementation. In addition, the thesis investigates implementation-level countermeasures, with particular focus on masking, and evaluates whether these techniques can prevent or complicate side-channel recovery. The following research questions were formulated during the preliminary project [Vik25] and are carried forward as the overall research questions for this thesis.

**RQ1** To what extent are multivariate post-quantum digital signature algorithms from the NIST second round susceptible to side-channel leakage?

**RQ2** Which specific suboperations within multivariate-based digital signature algorithms exhibit the highest susceptibility to side-channel leakage under practical attack scenarios?

**RQ3** How effective is a ChipWhisperer-based setup in capturing side-channel leakage from multivariate digital signature implementations?

**RQ4** What implementation-level countermeasures appear most effective in mitigating side-channel leakage in multivariate digital signature schemes, and how can their efficacy be evaluated through experimental analysis?

## 1.4 Related Work

One motivation for studying side-channel attacks on multivariate signature schemes in NIST's additional signatures process is that their implementation security has received limited attention. Many of the candidates are relatively recent proposals and only a limited number of side-channel studies have been published for these newer multivariate schemes. This makes implementation-level security analysis a rapidly developing research area. At the time the direction of this thesis was planned during the pre-project, only one paper directly targeting MAYO with side-channel analysis was available. This was the paper by Jendral and Dubrova, who presented the first evaluation of the resistance of the MAYO digital signature scheme to SCA [JD24]. They introduced two profiled, single-trace attacks, both exploiting leakage from modular multiplication operations and using deep-learning-assisted power analysis to recover information about the secret oil space. Both attacks target the bitsliced implementation of MAYO from the round-one setting, as this was the most recent implementation at the time. Since then, MAYO has moved towards a nibble-sliced implementation, where the internal arithmetic and leakage behavior may differ from the bitsliced version. They note that their second attack is also applicable to the nibble-sliced implementation, while the first one is not.

While the work on this thesis was already underway, a paper by Vishwaajith et al. [VGP+26] became available. This work studies non-profiled SCAs on multivariate digital signature implementations, with a particular focus on MAYO. They target a corresponding secret-dependent operation as the first attack of Jendral and Dubrova, but for the nibble-sliced implementation of MAYO, which is the implementation direction considered for the second-round version of MAYO. In contrast to the profiled bitsliced attack, the nibble-sliced attack does not rely on deep learning or a profiled leakage model. Instead, it uses Correlation Power Analysis (CPA) and exploits the fact that the same type of secret-dependent arithmetic operation is repeated many times during one execution. By using these repeated operations, the attack can recover entries of the secret oil matrix one by one from a single trace.

Vishwaajith et al. also place the attack in a broader multivariate setting, arguing that the same attack strategy is relevant not only for MAYO but also for other multivariate candidates. This supports the motivation for studying MAYO as part of a larger family of multivariate schemes, rather than as an isolated target. At the end of their paper, they point to masking as an interesting future research direction for protecting multivariate implementations against both profiled and non-profiled single-trace attacks.

A related line of work focuses on protecting multivariate signatures against side-channel attacks. Kundu et al. propose mUOV [KNK+25], a masked implementation of the Unbalanced Oil and Vinegar digital signature scheme. Their work identifies sensitive operations in UOV and introduces masking techniques for several critical computations. Since UOV and MAYO are both based on Unbalanced Oil and Vinegar (UOV), the masked UOV construction may be relevant when considering countermeasures for MAYO. The operation targeted in the attack of Vishwaajith et al. has a close counterpart in UOV, which makes their masking approach a natural starting point for evaluating whether similar techniques can protect MAYO.

This thesis builds on these works by reproducing and studying the non-profiled CPA attack on nibble-sliced MAYO in detail. At the time we started the reproduction work, no public attack code was available to us, so the attack methodology was implemented independently based on the paper description. In addition, we investigate the future direction suggested by Vishwaajith et al. by implementing masking for the target operation to evaluate whether share-wise masking mitigates the attack.

## 1.5 Contribution and Thesis Outline

This thesis contributes to the study of implementation security for multivariate post-quantum digital signatures by investigating power-based side-channel leakage in MAYO. The main contributions of this thesis are threefold. First, we provide a detailed methodology for adapting and evaluating a PQC implementation in a ChipWhisperer environment. Second, we provide a detailed explanation and independent reproduction of the non-profiled CPA attack on the nibble-sliced MAYO implementation described in [VGP+26]. Third, we implement and evaluate masking of the targeted operation using both two and three shares, and analyze whether this countermeasure prevents or complicates recovery under the considered attack model.

The thesis is organized as follows.

**Chapter 1** introduces the motivation, context, and research questions. We also summarize related work and present the main contributions.

**Chapter 2** provides the theoretical background for the thesis. We introduce digital signatures, Multivariate Public Key Cryptography (MPKC), MAYO, Side-Channel Attacks (SCA), Correlation Power Analysis (CPA), and relevant countermeasures.

**Chapter 3** describes the experimental methodology. We present the ChipWhisperer setup, define the attack scope, and explain the trace collection, processing, leakage analysis, and masking evaluation.

**Chapter 4** presents the experimental results. We analyze the collected traces, evaluate the CPA attack on MAYO, and present the masking results for two and three shares.

**Chapter 5** discusses the results and their limitations. We consider practical challenges, the transferability of the attack strategy, and implications for side-channel protection.

**Chapter 6** concludes the thesis by summarizing the main findings and answering the research questions.

## 1.6 Sustainability

This thesis relates to the UN Sustainable Development Goals (SDGs) [Uni26c] by addressing the long-term security and resilience of digital infrastructure. The most relevant goals are SDG 9 and SDG 16. SDG 9 [Uni26b] concerns resilient infrastructure and innovation, which is relevant because secure cryptographic mechanisms are a necessary part of reliable digital infrastructure. SDG 16 [Uni26a] concerns peaceful and inclusive societies, access to justice and accountable institutions, which is relevant because digital signatures support trust, integrity and accountability in digital systems.

The connection to SDG 9 lies in the role of cryptography as an enabling technology for digital infrastructure. Digital signatures are part of this foundation because they help protect authenticity and integrity in digital systems [Sma16, p. 217]. As current public-key mechanisms may become vulnerable to future quantum computers, the transition to post-quantum cryptography is important for maintaining resilient digital services over time [Nat17; BBD09]. However, this transition should not rely solely on the mathematical security of new algorithms. If post-quantum schemes leak secret-dependent information when implemented on real devices, they may fail to provide the security expected from them in practice [OG21]. Studying side-channel leakage in a post-quantum signature implementation is therefore relevant to understanding practical security challenges associated with resilient digital infrastructure.

The thesis also relates to SDG 16 through the role of digital signatures in supporting trust and accountability. Digital signatures help make digital actions verifiable by showing who produced a message, document or transaction, and whether it has been modified [Sma16, p. 217]. These properties may support accountable institutions and legal or administrative processes that increasingly rely on digital records. If future quantum computers weaken today's signature schemes, or if post-quantum replacements are insecure in practice, these trust mechanisms may be undermined. Efforts to evaluate and strengthen post-quantum digital signatures are therefore relevant to maintaining reliable and accountable digital institutions.

# Chapter 2
# Theory

In this chapter, we present the theoretical background required for the Side-Channel Attack (SCA) conducted in this thesis. We begin by introducing Digital Signature Algorithms (DSAs), before moving on to multivariate-based cryptography. We then describe the UOV scheme, which belongs to this family of cryptography and forms part of the foundation for MAYO, the DSA analyzed in this thesis. Subsequently, we introduce SCA in general, before focusing more closely on correlation power analysis and related concepts. Finally, we briefly mention selected side-channel countermeasures.

This theory chapter is closely related to the preliminary project [Vik25] that served as preparation for this master's thesis. Some sections therefore contain reused or adapted material from that work, with references provided to the original sources.

## 2.1 Digital Signature Algorithms

Digital signatures are an important cryptographic mechanism for creating trust in digital communication. They make it possible to verify that a message was produced by the claimed sender, in a way that is comparable to the role of handwritten signatures in physical documents [PPG24, p. 261]. However, digital signatures are based on public-key cryptography where each user has a private signing key and a related public verification key [WWK+24]. The signing key must remain secret, while the verification key can be distributed publicly. Since a valid signature can only be generated with the private signing key, a successful verified signature provides evidence that the message is linked to the owner of that key [PPG24, p. 303]. A digital signature scheme is defined by three core algorithms [WWK+24]:

**Key Generation** creates a key pair, consisting of a private signing key $sk$ and a public verification key $vk$.

**Signing** takes a message and the private signing key as input and produces a signature.

**Verification** takes the message, the signature and the public verification key as input, and determines whether the signature is valid.

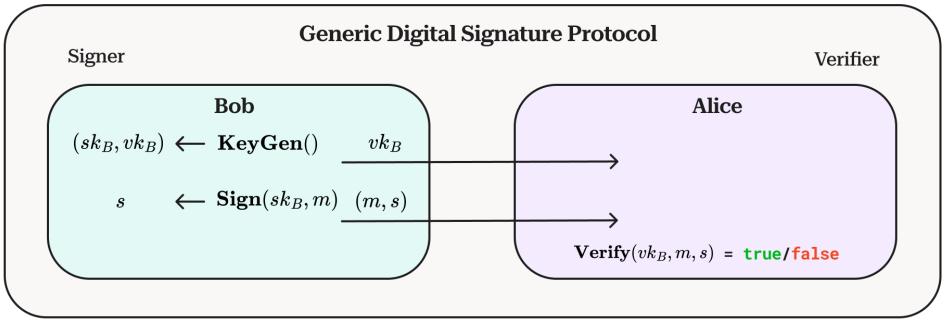



**Figure 2.1:** Generic digital signature protocol between two parties, Alice and Bob. Inspired by [PPG24, p. 303]

Following Figure 2.1, which illustrates a basic digital signature protocol involving Alice and Bob, Bob first generates a key pair $(sk_B, vk_B)$, and publishes his public key so that others, including Alice, can access it. To sign a message $m$, Bob applies the signing algorithm $\mathsf{Sign}(sk_B, m)$ to produce the signature $s$. Upon receiving the message with the attached signature, Alice verifies it using Bob's public key by applying the verification algorithm $\mathsf{Verify}(vk_B, m, s)$, which outputs either true or false. If the verification succeeds, Alice can be confident that the message was indeed generated by Bob and that it has not been altered during transmission.

**Security Services and Compromise of Digital Signature Algorithms**
Digital signatures must provide the following security properties to be considered secure [Sma16; PPG24, p. 217, 263]:

- **Authenticity**: Ensures that the message originates from the claimed sender. Since only the signer possesses the private signing key, a valid signature serves as proof of the sender's identity.
- **Integrity**: Guarantees that the message has not been modified after it was signed. Any alteration to the message will result in a verification failure due to the cryptographic binding between the message and the signature. Since the signature is computed over the message, or a hash of the message, even a small change in the message will cause the verification algorithm to reject the signature.
- **Non-repudiation**: Prevents the signer from denying having sent or signed a message.

A digital signature scheme is considered compromised if an adversary can undermine its core security guarantees, namely authenticity, integrity and non-repudiation. This may happen if the attacker recovers the signer's private signing key or is able to generate a valid signature for a new message without knowledge of the private signing key [Sma16, p. 217]. The most severe form of compromise is a total break, where the private signing key is recovered, enabling the attacker to sign any message. Weaker forms include selective forgery, where the attacker can forge a signature for a specific chosen message, and existential forgery, where the attacker can produce at least one valid signature for some new message of their choice [Sma16, p. 217].

Formally, a digital signature scheme is considered secure if it is Existentially UnForgeable under a Chosen-Message Attack (EUF-CMA) [Sma16; WWK+24]. Here, the adversary is adaptive, which means that it can adjust its queries based on previously obtained information. Under this definition, even if such an adaptive adversary obtains signatures on messages of their choice, it should remain computationally infeasible to forge a valid signature on any message [Sma16, p. 218].

## 2.2 Multivariate Public Key Cryptography

MPKC is a family of cryptographic schemes whose security relies on the assumed computational hardness of solving systems of nonlinear multivariate equations over finite fields [BBD09, p. 193]. In most multivariate schemes, these equations are quadratic due to efficiency reasons [DPS20, p. 9], and the underlying hard problem is therefore often referred to as the Multivariate Quadratic (MQ) problem. While Shor's algorithm [Sho97] can break widely used classical public-key schemes, such as RSA [RSA78], no efficient quantum algorithm is currently known for solving general MQ problems. For this reason, the family of multivariate-based cryptography is considered one of the primary candidates for post-quantum cryptography [DPS20, p. vii].

A central design challenge in multivariate-based cryptography is the creation of secure trapdoors [BBD09, p. 194]. In cryptography, a trapdoor refers to secret structural information embedded in a construction that appears random from the outside, but allows the legitimate key holder to efficiently solve a problem that would otherwise be computationally hard [Nat25; BBD09]. Such a structure is essential for the multivariate system to function as a cryptosystem. The underlying MQ problem, from which the trapdoor is built, is considered NP-hard [BBD09, p. 194]. However, once a trapdoor is introduced to make the system efficiently solvable for the legitimate user, the public polynomial system is no longer truly random. Consequently, the security of a specific MPKC scheme is not guaranteed solely by the NP-hardness of the general MQ problem [DPS20, p. vii]. The main design challenge is therefore to

construct a trapdoor that enables efficient legitimate use, while ensuring that the public system appears sufficiently random to adversaries.

Multivariate-based cryptography offers several practical advantages in addition to its assumed resistance to quantum attacks [DPS20, p. 23]. One important advantage is its efficiency, since the computations used in many multivariate schemes are typically simple arithmetic over small finite fields, which keeps the computational cost low. This makes many multivariate schemes efficient and suitable for constrained devices. Additionally, some multivariate signature schemes can produce very short signatures compared to other post-quantum alternatives, which is beneficial in bandwidth-limited environments. However, a notable drawback of multivariate-based cryptography is the large size of its public keys, which consist of systems of polynomial equations in many variables [DPS20, p. 23].

### 2.2.1 Unbalanced Oil and Vinegar

The UOV construction is an important trapdoor mechanism in multivariate-based digital signatures. Its relevance is reflected in NIST's call for additional post-quantum signature schemes, where four candidates that progressed to the third round are based on UOV [ABC+24; Nat26]. UOV is a modification of the original Oil and Vinegar (OV) scheme first introduced by Jacques Patarin in 1997 [OSSS97; BCH+23]. Both OV and UOV are single-field multivariate signature schemes, where computations are performed over a relatively small finite field, allowing efficient implementations [DPS20, p. 89].

The purpose of the OV mechanism is to introduce a trapdoor into a system of multivariate equations by hiding a solvable structure among the multivariate quadratic polynomials. This is achieved by separating the variables into two groups, namely oil variables and vinegar variables [DPS20, p. 90]. The construction is designed so that vinegar variables may interact with both oil and vinegar variables, while products between two oil variables are deliberately excluded [BBD09, p. 206]. As a result, once the vinegar variables are assigned fixed values, the remaining system becomes linear in the oil variables and can be solved by Gaussian elimination [DPS20, p. 91]. The secret partition between oil and vinegar variables functions as the trapdoor. An outside observer who only sees the public key should not be able to identify this structure, and the public key should appear as a random system of quadratic equations. The legitimate signer, however, can use the hidden partition to solve the system and generate valid signatures [DPS20, pp. 90-92].

#### Oil and Vinegar

The OV scheme can be described through the three core algorithms of a digital signature scheme, namely key generation, signing, and verification, as introduced in

Section 2.1. The following presentation of OV is based on the description by Ding et al. [DPS20, p. 90-92], the work on UOV by Kipnis, Patarin, and Goubin [KPG99], and the preliminary project [Vik25].

Let $\mathbb{F} = \mathbb{F}_q$ be a finite field. The parameters are defined as follows:
- $q$ denotes the number of elements in the field.
- $o$ denotes the number of oil variables.
- $v$ denotes the number of vinegar variables.
- $n = o + v$ denotes the total number of variables.
- $m$ denotes the number of equations, with $m = o$ in OV/UOV.

**Key Generation** During key generation, the public verification key and the private signing key for the OV scheme are constructed. The signer chooses a secret affine transformation $T : \mathbb{F}^n \to \mathbb{F}^n$ and defines a central map $F : \mathbb{F}^n \to \mathbb{F}^m$. The central map is a set of $m$ polynomials of the form

$$f_i(x_1, \ldots, x_n) = \sum_{j,k \in \mathcal{V}} \alpha_{j,k}^{(i)} x_j x_k + \sum_{\substack{j \in \mathcal{V} \\ k \in \mathcal{O}}} \beta_{j,k}^{(i)} x_j x_k + \sum_{j \in \mathcal{V} \cup \mathcal{O}} \gamma_j^{(i)} x_j + \delta^{(i)},$$

for $i = 1, \ldots, m$, where all coefficients, $\alpha$, $\beta$, $\gamma$ and $\delta$, are randomly chosen from $\mathbb{F}$.

The variables $x_1, \ldots, x_n$ are partitioned into two sets: $\mathcal{V} = \{1, \ldots, v\}$ and $\mathcal{O} = \{v+1, \ldots, n\}$, where $\mathcal{V}$ denotes the vinegar variables and $\mathcal{O}$ denotes the oil variables. The key structural property is that there are no products between two oil variables. That is, terms of the form $x_j x_k$ with $j, k \in \mathcal{O}$ do not occur.

Given this construction, the private signing key consists of the central map $F : \mathbb{F}^n \to \mathbb{F}^m$ and the secret affine transformation $T : \mathbb{F}^n \to \mathbb{F}^n$. The public verification key is obtained by composing these as $P = F \circ T$, resulting in $m$ quadratic polynomials in $n$ variables, where the oil and vinegar structure is hidden.

**Signing** A signature $s$ for a message $m$ is generated by first computing the hash of the target message $h(m) = y$. Then, the goal is to find a vector $\mathbf{x}$ such that $P(\mathbf{x}) = \mathbf{y}$, where $P$ is the public key. From the signer's point of view, this problem is reduced to finding the preimage under the central map $F$, for which the secret partition of the oil and vinegar variables is known. Due to this structure, the multivariate system reduces to a linear system in the oil variables once the vinegar variables are fixed. The signing procedure can be summarized in the following steps:

1. Assign random values to the vinegar variables.
2. Solve the resulting linear system $F(\mathbf{x}) = \mathbf{y}$ for the oil variables.
3. Compute the signature $\mathbf{s} = T^{-1}(\mathbf{x})$ to hide the structure.

**Verification** To verify the signature $\mathbf{s}$ on a message $m$, the verifier first computes the hash value $h(m) = \mathbf{y}$. The verifier then evaluates the signature using the public key to compute $\mathbf{y}' = P(\mathbf{s})$. If $\mathbf{y} = \mathbf{y}'$, the signature is accepted as valid.

The original OV scheme was later broken by a cryptanalytic attack proposed by Kipnis and Shamir in 1998 [KS98], which enabled recovery of the secret oil subspace and thereby allowed to forge signatures [DPS20, pp. 94-96]. Kipnis, Patarin, and Goubin introduced UOV to address this weakness by creating an imbalance between the number of oil and vinegar variables [KPG99]. While the original OV scheme used an equal number of oil and vinegar variables, $v = o$, the unbalanced variant instead uses more vinegar variables than oil variables, $v > o$ [DPS20, p. 109].

This modification prevents the Kipnis-Shamir attack from applying in the same way as in the balanced case, while preserving the central idea of the OV construction [DPS20, p. 109]. Increasing the number of vinegar variables significantly enlarges the search space, making it more difficult to identify the secret oil space. The complexity of the attack against UOV can be estimated as $q^{v-o-1}o^4$ [DPS20, p. 110], which shows that increasing $v$ improves resistance against this attack. However, the number of vinegar variables cannot be increased arbitrarily as larger values of $v$ may increase vulnerability to other types of attacks [DPS20, p. 110-115]. In addition, the size of the public key grows quadratically with the number of vinegar variables, while the signature grows linearly [DPS20, p. 110]. This introduces a trade-off between security and efficiency. A commonly suggested parameter choice is $v = 2o$ [DPS20, p. 110], which provides a balance between improved security and performance. In this sense, UOV preserves the efficient single-field signing structure of the original OV scheme, while strengthening it through increased asymmetry between the variable groups.

### 2.2.2 MAYO

MAYO is a post-quantum digital signature scheme belonging to the family of multivariate quadratic constructions. It was proposed by Ward Beullens in work presented at SAC 2021 [Beu22] and is one of the candidates that advanced to the second and now third round of the NIST additional signatures standardization process [ABC+24; Nat26]. MAYO is based on the Oil and Vinegar framework and builds on ideas from UOV while introducing structural modifications to address its main practical weakness, namely large public key sizes [BCC+25, p. 1].

As noted earlier, large public keys are a common drawback of multivariate-based cryptography. In traditional UOV, the public map must hide a secret oil space whose dimension equals the number of equations, $o = m$. This typically requires large parameters, resulting in public key sizes that limit the practicality of UOV as a general-purpose signature scheme [MAY]. MAYO mitigates this limitation by

reducing the size of the oil space, thereby allowing smaller public keys [BCC+25, p. 1-2]. However, a smaller oil space makes the ordinary UOV signing procedure overdetermined, which prevents the signer from efficiently solving the linear system. To overcome this, MAYO introduces a whipping mechanism, where the public map is expanded into a larger derived map. This enlarged structure provides a sufficiently large oil space for signing, while still allowing the scheme to benefit from reduced public key sizes [BCC+25, p. 1-3].

### MAYO Core Construction

The reduced oil space $\mathbf{O}$, with dimension $o < m$, on which $P$ evaluates to zero, makes the standard OV signing procedure insufficient, since the resulting linear system in the oil variables contains fewer unknowns than equations and may therefore fail to admit a solution. The whipping mechanism resolves this by transforming the original public map $P$ into an expanded map $P^*$, where the oil space is enlarged, and the signing equation becomes solvable with the standard OV procedure [BCC+25, p. 2]. The expanded map is constructed by combining several copies of the original map $P$, its differential $P'(x, y) = P(x + y) - P(x) - P(y)$ and fixed public matrices $E_{ij}$ to bind the different components together [BCC+25, p. 2]. The public map is defined as:

$$P^*(x_1, \ldots, x_k) = \sum_{i=1}^{k} E_{ii} P(x_i) + \sum_{i=1}^{k} \sum_{j=i+1}^{k} E_{ij} P'(x_i, x_j).$$

The parameter $k$, called the whipping parameter, is chosen so that the oil space in the expanded construction has dimension $ko > m$ [BCC+25, p. 2]. This makes sure that the oil space $O^k$ is large enough for the signer to find solutions to the linear system and sample signatures [MAY].

### MAYO algorithms

The functionality of the MAYO signature scheme is structured around five algorithms, namely MAYO.CompactKeyGen, MAYO.ExpandSK, MAYO.ExpandPK, MAYO.Sign, and MAYO.Verify [BCC+25, p. 9]. The expansion algorithms MAYO.ExpandSK and MAYO.ExpandPK reconstruct the algebraic structures required for signing and verification. For compliance with the NIST API, MAYO implements the three algorithms MAYO.API.keypair, MAYO.API.sign, and MAYO.API.sign_open [BCC+25, p. 13]. Within these, MAYO.ExpandSK is used during signing, while MAYO.ExpandPK is used during verification.

To describe the main procedures without reproducing the full specification, Algorithms 2.1, 2.2, and 2.5 show simplified versions of the three core algorithms of DSAs, where low-level details are omitted for readability. The complete algorithmic descriptions can be found in the MAYO round-two specification [BCC+25]. Furthermore,

Algorithms 2.3 and 2.4 are included to clarify the signing path, where MAYO.API.sign invokes MAYO.ExpandSK before calling MAYO.Sign.

**Algorithm 2.1** MAYO.CompactKeyGen()

**Output:** Compact secret key csk, compact public key cpk
*// Pick a secret seed*
Sample a random secret seed $\mathsf{seed}_{sk}$
*// Derive the public seed and secret oil-space matrix*
Use SHAKE256($\mathsf{seed}_{sk}$) to derive $\mathsf{seed}_{pk}$ and $\mathbf{O}$
*// Derive the first two parts of the public map*
Generate $\{\mathbf{P}_i^{(1)}\}_{i\in[m]}$ and $\{\mathbf{P}_i^{(2)}\}_{i\in[m]}$ from $\mathsf{seed}_{pk}$ using AES-128-CTR
*// Compute the third part of the public map*
**for** $i = 0$ to $m-1$ **do**
    Compute $\mathbf{P}_i^{(3)} = \mathrm{Upper}(-\mathbf{O}^\top\mathbf{P}_i^{(1)}\mathbf{O} - \mathbf{O}^\top\mathbf{P}_i^{(2)})$
*// Output compact keys*
Set $\mathsf{cpk} = \mathsf{seed}_{pk} \,\|\, \textsc{Encode}_{P^{(3)}}(\{\mathbf{P}_i^{(3)}\}_{i\in[m]})$
Set $\mathsf{csk} = \mathsf{seed}_{sk}$
**return** (cpk, csk)

**Algorithm 2.2** MAYO.Sign(esk, $M$)

**Input:** Expanded secret key esk, message $M$
**Output:** Signature sig
**Constant:** $\mathbf{E} \in \mathbb{F}^{m\times m}$
Decode $\mathbf{O}$, $\{\mathbf{P}_i^{(1)}\}_{i\in[m]}$, and $\{\mathbf{L}_i\}_{i\in[m]}$ from esk
Compute the message digest using SHAKE256($M$)
Derive salt and target vector $\mathbf{t}$
*// Attempt to find a preimage for* $\mathbf{t}$
**for** ctr $= 0$ to $255$ **do**
    Derive vinegar variables $\mathbf{v}_0, \ldots, \mathbf{v}_{k-1}$
*// Build the linear system* $\mathbf{Ax} = \mathbf{y}$
    Initialize $\mathbf{A}$ and set $\mathbf{y} \leftarrow \mathbf{t}$
    **for** $i = 0$ to $k-1$ **do**
        Compute coefficients of $\mathbf{A}$ from terms involving $\mathbf{v}_i$ and $\{\mathbf{L}_j\}_{j\in[m]}$
        Update $\mathbf{y}$ using quadratic vinegar terms involving $\{\mathbf{P}_a^{(1)}\}_{a\in[m]}$
*// Try to solve the system*
    Solve $\mathbf{Ax} = \mathbf{y}$
    **if** a solution $\mathbf{x}$ is found **then**
        **break**
*// Construct the signature vector*
**for** $i = 0$ to $k-1$ **do**
    Construct $\mathbf{s}_i \leftarrow \{\mathbf{v}_i + \mathbf{O}\mathbf{x}_i \,\|\, \mathbf{x}_i\}$
**return** sig $\leftarrow \textsc{Encode}(\mathbf{s}) \,\|$ salt

**Algorithm 2.3** MAYO.ExpandSK(csk)

**Input:** Compact secret key csk
**Output:** Expanded secret key esk

Extract $\mathsf{seed}_{sk}$ from csk
Derive $\mathsf{seed}_{pk}$ and the secret matrix $\mathbf{O}$ from $\mathsf{seed}_{sk}$
Reconstruct $\{\mathbf{P}_i^{(1)}\}_{i\in[m]}$ and $\{\mathbf{P}_i^{(2)}\}_{i\in[m]}$ from $\mathsf{seed}_{pk}$
**for** $i = 0$ to $m-1$ **do**
    Compute $\mathbf{L}_i \leftarrow (\mathbf{P}_i^{(1)} + \mathbf{P}_i^{(1)\top})\mathbf{O} + \mathbf{P}_i^{(2)}$
Encode the matrices $\{\mathbf{L}_i\}_{i\in[m]}$
Set esk $\leftarrow \mathsf{seed}_{sk} \parallel \mathbf{O} \parallel \mathbf{P}^{(1)} \parallel \textsc{Encode}_L(\{\mathbf{L}_i\}_{i\in[m]})$
**return** esk

**Algorithm 2.4** MAYO.API.Sign($M$, sk)

**Input:** Secret key sk, message $M$
**Output:** Signed message sm
*// Expand the secret key*
Set esk $\leftarrow$ MAYO.ExpandSK(sk)
*// Produce the signature*
Set sig $\leftarrow$ MAYO.Sign(esk, $M$)
*// Return the signed message*
**return** sm $\leftarrow$ sig $\parallel M$

**Algorithm 2.5** MAYO.Verify(epk, $M$, sig)

**Input:** Expanded public key epk, message $M$, signature sig
**Output:** Valid or invalid
**Constant:** $\mathbf{E} \in \mathbb{F}^{m\times m}$
*// Decode the expanded public key*
Decode $\{\mathbf{P}_i^{(1)}\}_{i\in[m]}$, $\{\mathbf{P}_i^{(2)}\}_{i\in[m]}$, and $\{\mathbf{P}_i^{(3)}\}_{i\in[m]}$ from epk
*// Decode the signature*
Extract salt from sig
Decode the signature vector $\mathbf{s}$
*// Hash the message and derive the target vector*
Compute the message digest using SHAKE256($M$)
Derive the target vector $\mathbf{t}$ from the message digest and salt
*// Evaluate the public map on the signature*
Compute $\mathbf{y} \leftarrow \mathbf{P}^*(\mathbf{s})$ using $\mathbf{P}^{(1)}$, $\mathbf{P}^{(2)}$, $\mathbf{P}^{(3)}$, and $\mathbf{E}$
*// Accept if the evaluation matches the target*
**if** $\mathbf{y} = \mathbf{t}$ **then**
    **return** valid
**else**
    **return** invalid

**Key generation**: Key generation in MAYO [BCC+25, p. 9] constructs a secret oil space together with a corresponding public map that vanishes on this space [BCC+25, p. 2], and outputs compact representations of the public and secret keys. Following the MAYO key generation algorithm [BCC+25, p. 9], simplified in Algorithm 2.1, the procedure begins by sampling a random secret seed `seed_sk`, from which a public seed `seed_pk` is derived. The secret oil space is then derived as a matrix $\mathbf{O}$ using SHAKE256 [Nat15]. The public map is defined through three sets of matrices. The first two sets, $\mathbf{P1}$ and $\mathbf{P2}$, are randomly generated using AES-128 in counter mode [BCC+25, p. 9]. The third set, $\mathbf{P3}$, is then computed so that the resulting quadratic map vanishes on the oil space defined by $\mathbf{O}$. The public key consists of the public seed `seed_pk` together with an encoded representation of $\mathbf{P3}$, while the secret key is given by the original secret seed `seed_sk` [BCC+25, p. 9]. This compact representation allows the full key material to be deterministically reconstructed when needed.

**Signing**: The goal of the signing procedure is to compute a preimage of a message-dependent target vector under the expanded map $P^*$. Due to the reduced oil space, this requires working in the enlarged system produced by the whipping mechanism. At the API level, MAYO.API.sign 2.4 first expands the secret key using MAYO.ExpandSK 2.3, and then invokes MAYO.Sign 2.2 to compute the signature [BCC+25, p. 13].

The purpose of the secret key expansion algorithm MAYO.ExpandSK, shown in Algorithm 2.3, is to reconstruct the full secret material from the compact secret seed [BCC+25, p. 10]. As part of this process, the matrices $\mathbf{L}_i$ are computed, which are used to efficiently build the linear system during signing. They are given by $\mathbf{L}_i = (\mathbf{P}_i^{(1)} + \mathbf{P}_i^{(1)\top})\mathbf{O} + \mathbf{P}_i^{(2)}$, where $\mathbf{P}_i^{(1)}$ and $\mathbf{P}_i^{(2)}$ are public matrices derived from the public seed, while $\mathbf{O}$ is the secret oil space matrix [BCC+25, p. 10]. Thus, these matrices capture the interaction between the public map and the secret structure.

The core signing procedure MAYO.Sign, is responsible for computing a signature by finding a preimage of the target vector under the expanded map. Concretely, the goal is to determine a vector $\mathbf{s}$ such that $P^*(\mathbf{s}) = \mathbf{t}$, where $\mathbf{t}$ is derived from the message and salt [BCC+25, p. 2]. The procedure can be summarized in three main steps, following MAYO.Sign from the MAYO specification [BCC+25, p. 11], presented in simplified form in Algorithm 2.2:

1. **Hash computation**: The signing algorithm is given the expanded secret key and the message to be signed as input. The signer first computes a message hash using SHAKE256. A salt is then derived, and the target vector $\mathbf{t}$ is computed from the hash and the salt. This target vector is the value that the signature must evaluate to under $P^*$.

2. **Preimage search**: Since a solution may not exist for a given choice of variables, the signing algorithm repeats the following steps until a valid solution is found. First, the signer samples random vinegar variables $\mathbf{v}_0, \ldots, \mathbf{v}_{k-1}$ and the linear system is initialized with $\mathbf{y} \leftarrow \mathbf{t}$. Intuitively, substituting the vinegar variables into the public map gives $P(\mathbf{v}+\mathbf{o}) = P(\mathbf{v}) + P'(\mathbf{v},\mathbf{o}) + P(\mathbf{o})$. Here $P'(\mathbf{v},\mathbf{o})$ is linear in the oil variables, $P(\mathbf{v})$ is constant and $P(\mathbf{o}) = 0$ by the construction of the oil space. Thus, after the vinegar variables have been fixed, the remaining equations become linear in the unknown oil variables. Using this structure, together with the precomputed matrices $\mathbf{L}_i$ and the coupling matrix $\mathbf{E}$, the signer constructs a linear system of the form $\mathbf{Ax} = \mathbf{y}$, where the unknowns $\mathbf{x}$ are the oil coordinates across all components. These coordinates are later used to construct the oil-space contribution in the final signature vector. This system is then solved using Gaussian elimination. If the resulting system has no solution, the signer samples a new set of vinegar variables and repeats the procedure.

3. **Signature construction**: Once a valid solution is found, the oil-space coordinates $\mathbf{x}$ are combined with the vinegar variables and the secret matrix $\mathbf{O}$ to construct the signature vector. For each component, this is done as $\mathbf{s}_i = \{\mathbf{v}_i + \mathbf{Ox}_i \parallel \mathbf{x}_i\}$. The encoded vector $\mathbf{s}$ and the $\mathsf{salt}$ is then returned as the final signature $\mathsf{sig}$.

**Verification**: The verification algorithm checks whether the signature $\mathbf{s}$ evaluates to the correct target vector under the whipped public map. The verifier uses the public seed to reconstruct the public structures needed for verification and derives the target vector $\mathbf{t}$ from the message and the salt contained in the signature. The signature is then accepted by the verifier only if $P^*(\mathbf{s})$ equals the target vector $\mathbf{t}$ [BCC+25, p. 12].

### Data Representation

From round 1 to round 2 of the NIST process, the MAYO reference implementation has transitioned from a bitsliced to a nibble-sliced data representation [BCC+25, p. 3]. This change affects how data are stored and how arithmetic over the finite fields is implemented. Bitsliced and nibble-sliced refer to two different data representation and organization techniques. A bitsliced representation stores data by grouping together bits from the same position across several field elements. Instead of storing each element as a separate unit, the bits are spread across registers so that many elements can be processed in parallel using logical bit operations [BCC+24]. In contrast, a nibble-sliced representation stores field elements in a compact form using four-bit blocks, called nibbles. Since MAYO operates over $\mathbb{F}_{16}$, each element can be

represented using four bits, allowing two elements to be stored in a single byte. This layout enables efficient use of modern CPU features [BCC+24].

Although both formats rely on logical operations, they differ in how they handle more expensive operations such as multiplication [BCC+24]. In a bitsliced implementation, multiplication is performed through sequences of bitwise instructions. Nibble-slicing, on the other hand, enables lookup-based techniques. This is supported through the Method of the Four Russians (M4R) [Bar09; ADKF70], which replaces costly field multiplications with cheaper table lookups. The shift to nibble-slicing was motivated by the observation that bitslicing, while useful for some microcontrollers, limited the performance on platforms with more advanced arithmetic capabilities. The nibble-sliced format enables efficient use of shuffle instructions on AVX2 and NEON, while also supporting faster matrix multiplication on Arm Cortex-M4 through M4R [BCC+24].

## 2.3 Side-Channel Attacks

SCAs exploit information that is unintentionally leaked by a cryptographic implementation during execution [OG21, p. 1]. Unlike classical cryptanalysis, which targets weaknesses in the mathematical design of an algorithm, side-channel analysis focuses on how the algorithm behaves when it is implemented. Even if a cryptographic scheme is secure from a mathematical point of view, its implementation may reveal information through side channels. A side channel is an unintended communication path through which information about the internal state of the device can be observed [OG21, p. 10]. Common examples include execution time, power consumption, and electromagnetic radiation, where the latter two are the most exploited ones [OG21, p. 10]. In this thesis, the relevant source of leakage is power consumption measured during execution of the signing operation of a DSA.

In a power analysis attack, the adversary records the power consumption of the target device while it processes secret-dependent data [OG21, p. 1]. These measurements are represented as power traces, which show the variation in power consumption over time during one execution of the targeted operation. Power traces may be collected across multiple executions, or from a single execution if the same operation is repeated and the corresponding leakage can be isolated in sub-traces. The latter is referred to as a single-trace attack [VGP+26]. By analyzing power traces, the attacker can apply different analysis techniques to determine which secret value hypothesis best explains the observed leakage [OG21, p. 1].

Side-channel attacks can be classified according to the adversary's capabilities. A common distinction is made between non-profiled and profiled attacks [DRC+25; OG21]. In a non-profiled attack, the adversary attacks the targeted device directly

without first characterizing an identical device. Such attacks often rely on general assumptions about the leakage behavior of the implementation by using a standard leakage model [OG21, p. 10-11]. Non-profiled attacks include simple, differential, correlation, and non-profiled deep learning-based attacks [DRC+25, p. 6]. In contrast, a profiled attack assumes that the adversary has access to a device that is identical, or very similar to the target. This device is then used in a profiling phase to learn how the implementation leaks information before the actual attack is performed on the target device [DRC+25, p. 6].

### 2.3.1 Hamming Weight

To perform a non-profiled statistical SCA, such as Differential Power Analysis (DPA) or CPA, the attacker needs a way to connect the internal values processed by the device with the physical leakage that can be measured externally. A leakage model provides this link by relating the device's data-dependent intermediate computations to observable side-channel information, such as power consumption [VO22, p. 318]. Different leakage models may be used depending on the target architecture, the implementation and the attacker's knowledge.

One commonly used model in non-profiled power analysis is the Hamming Weight (HW) leakage model. The HW of a value $z$, denoted $HW(z)$, is defined as the number of bits set to one in the binary representation of $z$. For example, the binary value 1011 has HW 3. The HW leakage model assumes that the power consumed by a device is related to the HW of the value being processed [OG21, p. 11]. This means that values with more bits set to one are assumed to cause a different power consumption than values with fewer bits set to one.

The relationship between the measured leakage and the sensitive value can be modeled as

$$L_q = a \cdot \mathrm{HW}(z) + c + n_q, \tag{2.1}$$

where $L_q$ is the leakage observed at sample point $q$, $z$ is the sensitive intermediate value, $a$ and $c$ are constant parameters, and $n_q$ represents noise [OG21, p. 11]. This model is especially useful when the leakage is related to values on a pre-charged bus, for instance when Random Access Memory (RAM) is addressed [OG21, p. 11] during read or write operations.

### 2.3.2 Correlation Power Analysis

CPA is a non-profiled side-channel attack method that uses statistical correlation to recover secret information from a cryptographic implementation. The method relies on a leakage model to transform hypotheses about secret-dependent intermediate values into predicted leakage values, so that the hypotheses are represented in a

form that can be compared with the measured power consumption [OG21, p. 10, 23]. For each hypothesis, the predicted leakage is compared with the measured power using the **Pearson correlation coefficient** [Pea01]. This coefficient measures the strength of a linear relationship between two variables and takes values between -1 and 1 [VO22, p. 312]. A value close to 1 indicates a strong positive linear relationship, meaning that the two variables tend to increase and decrease together. A value close to -1 indicates a strong negative linear relationship meaning that they behave in opposite directions, while a value close to 0 indicates little or no linear relationship [VO22, p. 312]. In CPA, the two variables are the computed hypothetical leakage predicted by the leakage model and the measured power consumption at a specific point in time [VO22, p. 315]. The correct secret hypothesis is expected to produce predicted leakage that agrees more closely with the measured leakage, and therefore give a higher correlation. The CPA process can be described in four main steps [OG21]:

1. **Leakage modeling**: The attacker selects a leakage model and constructs hypotheses for the targeted secret-dependent intermediate values

2. **Trace collection**: Power traces are collected from executions of the targeted operation.

3. **Statistical distinguishing**: The Pearson correlation coefficient is used as a statistical distinguisher to compare the hypothetical leakage with the measured power consumption

4. **Key recovery**: The candidate that produces the highest correlation score is selected as the most likely secret value.

For a set of $N$ traces, the Pearson correlation coefficient can be computed for each key hypothesis $k$ and each sample point $q$. It compares the hypothetical leakage values $H_k$ predicted for hypothesis $k$ with the measured power samples $T_q$ at sample point $q$. This can be written as [OG21; VO22]:

$$\rho_{k,q} = \frac{\sum_{i=1}^{N} \left(H_{i,k} - \overline{H_k}\right)\left(T_{i,q} - \overline{T_q}\right)}{\sqrt{\sum_{i=1}^{N} \left(H_{i,k} - \overline{H_k}\right)^2} \sqrt{\sum_{i=1}^{N} \left(T_{i,q} - \overline{T_q}\right)^2}}.$$

Here, $H_{i,k}$ denotes the hypothetical leakage for trace $i$ under key hypothesis $k$, while $T_{i,q}$ denotes the measured power consumption of trace $i$ at sample point $q$. The terms $\overline{H_k}$ and $\overline{T_q}$ denote the corresponding mean values over all $N$ traces.

**Secret-dependent operations in MAYO**

For a SCA to succeed in recovering sensitive information from an algorithm, the measured leakage must correspond to a point in time where secret-dependent information is processed. The implementation of a DSA can be studied to determine where such operations occur. For MAYO, several secret-dependent operations have already been identified in [KNK+25; VGP+26], as summarized in Table 2.1:

**Table 2.1:** Secret-dependent operations in MAYO, based on [KNK+25; VGP+26].

| Computation | Secret value | Where it occurs |
|---|---|---|
| $\mathbf{L}_i \leftarrow (\mathbf{P}_i^{(1)} + \mathbf{P}_i^{(1)\top})\mathbf{O} + \mathbf{P}_i^{(2)}$ | $\mathbf{O}$ | Signing, during secret key expansion, ExpandSK. |
| $\mathbf{v}^\top \mathbf{P}_i^{(1)} \mathbf{v}$ | $\mathbf{v}$ | Signing, during evaluation with the selected vinegar variables. |
| $\mathbf{s}_i = \{\mathbf{v}_i + \mathbf{O}\mathbf{x}_i \parallel \mathbf{x}_i\}$ | $\mathbf{O}$, $\mathbf{v}_i$ | Signing, during construction of the signature vector. |

These operations occur during the MAYO signing procedure and may be used as intermediate values for computing predicted leakage, for example through their HWs, which can then be compared with the measured traces using the Pearson correlation coefficient.

## 2.4 Side-Channel Protection

The purpose of side-channel countermeasures is to reduce the exploitable relationship between secret data and physical leakage [OG21, p. 35]. In general, countermeasures try to make the measured leakage less useful and harder to obtain for the attacker, either by randomizing the sensitive data being processed or by making the leakage harder to align across traces. Two important categories of countermeasures are masking and hiding [OG21, p. 35].

### 2.4.1 Masking

Masking is a countermeasure that protects sensitive values by randomizing their representation during computation [OG21, p. 37]. The main idea is to break the direct dependency between a sensitive variable and the corresponding physical leakage. Instead of processing the sensitive value directly, the implementation processes several randomized shares, which only allows the original value to be recovered when all shares are combined [OG21, p. 38].

Masking can be implemented at an algorithmic level, where a sensitive value is split into $d+1$ shares, where $d$ denotes the masking order [OG21, p. 37]. The masking order determines the number of shares and the theoretical resistance against higher-order side-channel attacks. A higher-order attack refers to an attack that combines leakage from multiple shares, instead of relying on the leakage of a single sensitive variable [OG21, p. 27]. To defeat a $d$-th order masking scheme, the attacker must therefore typically combine leakage information from $d+1$ shares simultaneously. For example, for $d = 1$, the sensitive value is split into two shares. This first-order masking protects against first-order attacks, where the adversary considers the leakage from only one share at a time. Recovering information about the masked value therefore requires combining leakage from both shares, corresponding to a second-order attack.

**Boolean masking** is a common form of masking where a sensitive value $Z$ is split into shares using the XOR operation [OG21, p. 38]. Following the explanation of Boolean masking in [OG21, p. 38] we have that for a masking order $d$, the implementation generates $d$ random masks $R_1, \dots R_d$. The first share is computed as $Z_0 = Z \oplus R1 \oplus \cdots \oplus R_d$ from these random shares. The remaining shares are the random masks themselves $Z_1 = R_1, Z_d = R_d$, which allows the original sensitive value to be reconstructed by XORing all shares:

$$Z = \bigoplus_{i=0}^{d} Z^{(i)}$$

Since each individual share is randomized, observing the leakage from only one share should not reveal the original sensitive value [OG21, p. 38]. However, masking does not necessarily remove all leakage but instead forces the attacker to perform a higher-order attack.

### 2.4.2 Hiding

Hiding is another class of side-channel countermeasures. While masking changes the representation of sensitive data, hiding aims to make the physical leakage harder to exploit without necessarily changing the data itself [OG21, p. 35]. The purpose is to reduce the attacker's ability to identify, align or distinguish the leakage associated with sensitive operations. Side-channel attacks often rely on the assumption that the same operation occurs at the same time in each trace. If this alignment is disturbed, the statistical relationship between the measured traces and the attacker's predictions becomes weaker [OG21, p. 36]. Therefore, many hiding techniques work in the time dimension, trying to desynchronize the traces or make the execution order less predictable. Three techniques for hiding in the time-dimension are [OG21, p. 36]:

- **Insertion of dummy operations**: This technique adds no-operations or operations with dummy variables to the execution flow, which do not contribute

to the actual cryptographic computation but may disturb the timing of sensitive operations and make the traces harder to analyze.
- **Jittering**: Random delays are inserted during execution, which makes it more difficult for an attacker to align traces accurately.
- **Shuffling**: Independent operations are executed in a randomized order. Instead of processing data in a fixed predictable sequence, the implementation changes the order from one execution to another.

# Chapter 3
# Methodology

In this chapter, we describe the experimental setup and methodology used to reproduce the MAYO attack introduced by Vishwaajith et al. [VGP+26]. Hereafter, we refer to their work as `SCA-MQDSA`. Since no public implementation was available, our methodology is based on an independent interpretation of the attack description provided in `SCA-MQDSA` and its translation into our ChipWhisperer setup. The chapter therefore also includes our experience in adapting a PQC algorithm to this hardware setting, with practical observations on trace acquisition and processing.

We begin by presenting the hardware platform and the firmware organization used to run the MAYO implementation. We then define the scope of the reproduced attack, including its assumptions and simplifications. After this, we explain how power traces are acquired and processed, before describing the steps required to model the leakage and carry out the analysis. Finally, we evaluate masking as a countermeasure to mitigate the attack. These steps correspond to the five sections shown in Figure 3.1: Experimental Setup, Attack Scope, Data Collection, Leakage Analysis and Countermeasure Evaluation.

The work is carried out in two stages. First, selected parts of the attack are studied in isolation on the SAM4S platform, since it does not provide sufficient memory to execute the full target function. The insights gained from this initial phase are then transferred to the STM32F4 platform, where the methodology is applied to the full target function.


**Figure 3.1:** Methodology used to reproduce the attack on MAYO and evaluate countermeasures.

## 3.1 Experimental Setup

The experiments are conducted using SAM4S [Newd] and STM32F4 [Newc] target boards, each mounted on a CW313 adapter board. Power traces are captured using a ChipWhisperer Husky [Newa]. Communication between the host computer and the target device is implemented using the SimpleSerial protocol [Newj]. An overview of the entire experimental environment is presented in Figure 3.2.

### 3.1.1 ChipWhisperer

ChipWhisperer is an open-source toolchain for hardware security research developed and maintained by NewAE Technology Inc [Newg]. It is designed both for learning about side-channel attacks on embedded devices and for evaluating the resistance of cryptographic implementations against such attacks. Its main focus is on side-channel and fault attacks, in particular power analysis and glitching. The ChipWhisperer toolchain has four integrated layers of open-source components, which consist of capture hardware with target boards, target-side firmware, host-side analysis software, and Jupyter Notebook tutorials [Newg]. This makes it well suited both for educational purposes and for experimental work in side-channel analysis.

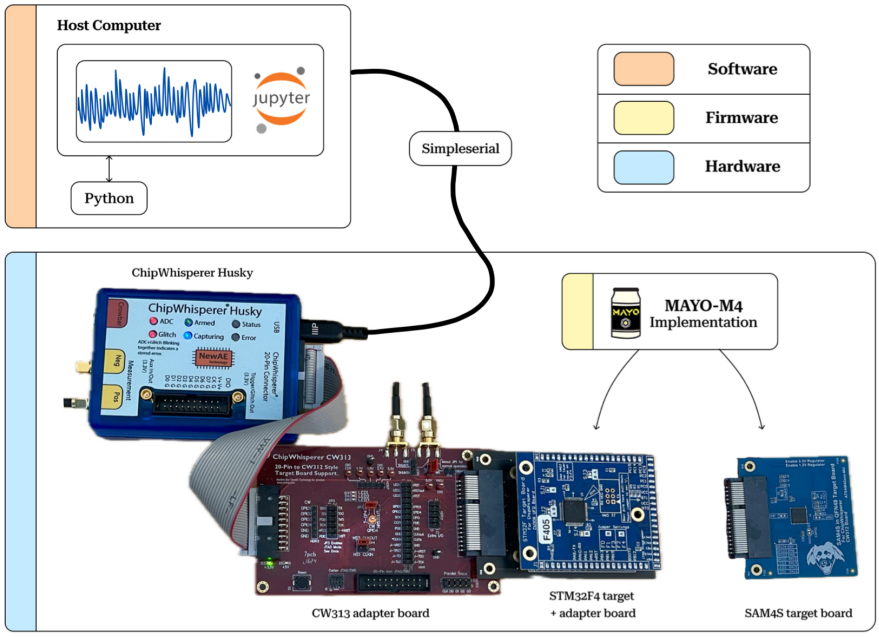



**Figure 3.2:** ChipWhisperer setup illustrating the software, firmware, and hardware layers of the experimental environment.

The distinction between hardware, firmware, and software in the ChipWhisperer architecture is useful to structure the experimental setup. Hardware refers to the physical components of the measurement setup. Firmware refers to code that is embedded in, or closely tied to, a hardware device and supports core functions such as start-up, execution control, and communication with other components [FS]. Software, by contrast, refers to the broader set of instructions used to run a computing system [FS] and here primarily denotes the host-side environment used to control the setup and analyze the results. Within this framework, the algorithm implementation is treated as firmware once it has been compiled and flashed onto the target. Accordingly, the original MAYO codebase may be viewed as software at the source-code level, but as part of the target firmware once it has been adapted to the ChipWhisperer build system and flashed onto the target board. In the following, the experimental setup is described according to this division into hardware, firmware, and software.

### Hardware

Figure 3.2 provides an overview of the experimental setup. The hardware layer consists of a ChipWhisperer Husky capture device together with SAM4S and STM32F4 target boards. A CW313 adapter board is used as an interposer between the ChipWhisperer 20-pin connector and the target board. For the STM32F4 target board, a CW308-to-CW312 breakout board is additionally used to connect the target to the CW313 board. The Husky is an open-source capture device and the successor to the CW1173 ChipWhisperer-Lite [Newa]. Compared to earlier ChipWhisperer capture hardware, it provides several features that are advantageous for side-channel analysis, including improved analog capture capabilities, larger sample buffers, and support for streaming mode [Newa].

Streaming mode allows for longer captures that are not limited by the device's sample buffer, as samples from the Analog-to-Digital Converter (ADC) are continuously transferred to the host while the trace is being recorded [Newf]. This is particularly useful in single-trace analysis, where traces may become too long to fit within the ordinary sample buffer. However, this comes at the cost of a lower maximum sampling rate, meaning that fewer samples are captured per clock cycle.

The target hardware consists of two different microcontroller boards. The first is the CW312T-SAM4S target with 128 KB of flash and 64 KB of SRAM [Newd]. The second is an STM32F4 target from the CW308 family, with 1 MB of flash and 192 KB of SRAM [Newc]. Both targets are based on Arm Cortex-M4-class microcontrollers, but they differ substantially in available memory.

Initial measurements were collected on both targets in order to compare how clearly relevant operations could be identified in the resulting traces. In these experiments,

the SAM4S traces appeared cleaner, and their internal structure was easier to recognize visually, whereas the STM32F4 traces were more difficult to interpret due to higher noise levels. For this reason, the SAM4S provided a useful starting point for the initial leakage analysis. Within this thesis, the two platforms are therefore used for different purposes. The SAM4S is used in the initial experimental phase, where its cleaner traces provide a more suitable basis for controlled leakage analysis. The STM32F4 is used for experiments on the full target function, as its larger available memory allows the complete MAYO computation to be executed.

**Communication between hardware components**
Communication in the experimental setup involves the host computer, the capture device, and the target. The host computer acts as the central control point. Through the ChipWhisperer Python API, it configures the Husky, sends commands to the target, arms the scope, initiates trace capture, and processes the resulting traces. The capture device serves both as a measurement instrument and as an intermediary between the host and the target, while the target receives commands and executes the requested computation [New25].

Communication between the host and the target is implemented using the SimpleSerial protocol [Newj]. The host computer issues a command through the ChipWhisperer Python interface. The capture device transmits this command over Universal Asynchronous Receiver-Transmitter (UART) [VO22, p. 46] to the target, where the packet is decoded and the corresponding command is processed. The target may then return data, followed by an acknowledgement packet [Newj].

### Firmware

The ChipWhisperer documentation includes a guide for custom firmware on ChipWhisperer targets [Newb]. The experimental firmware setup in this thesis follows this workflow closely, with the aim of capturing power traces during execution of the MAYO signing algorithm on a ChipWhisperer target board.

The custom firmware guide describes a firmware project as consisting of four main parts: the main project files, the Hardware Abstraction Layer (HAL), common crypto implementations such as AES, and the SimpleSerial folder [Newb]. The main project files include the application-specific source code and the project Makefile, while the HAL provides platform-specific code for tasks such as clock setup and serial communication. The crypto folder contains reusable cryptographic implementations, and the SimpleSerial folder provides the target-side code for the SimpleSerial protocol [Newb].

The MAYO implementation itself is the cryptographic algorithm to be executed on the target. To use it in the experiments, it must be integrated into a ChipWhisperer

firmware project, compiled with the appropriate platform settings, and linked against the relevant HAL and SimpleSerial components. In the context of this thesis, this means that the MAYO source code was adapted to the ChipWhisperer firmware structure so that it could be compiled and flashed to either the SAM4S or the STM32F4 target.

Within the `ChipWhisperer` repository [New26], the required HAL files, crypto and the target-side SimpleSerial implementation are already available. These resources are located in the `firmware/mcu` directory, which contains both platform-specific HAL folders and example projects. Therefore, the only part that must be created for this setup is the project-specific part. The firmware setup can be divided into six steps:

1. **Clone the ChipWhisperer Repository** The first step was to clone the ChipWhisperer Repository [New26] from GitHub. The HAL, SimpleSerial implementation, and build infrastructure are already provided in the repository and will be linked to the specific project in later steps.

2. **Create a project folder for the custom firmware** A new firmware project folder, `simpleserial-mayo1nssam4s`, was created under `firmware/mcu` in the cloned ChipWhisperer repository. In this repository name, `ns` is short for nibble-sliced.

3. **Add the source code to the project folder** The complete MAYO implementation was copied into the project folder. In this setup, the dedicated crypto folder is not used, since the implementation is not intended for reuse across multiple projects.

4. **Create the project-specific source file** The project-specific source file `simpleserial-mayo1nssam4s.c` was created within the project folder. This file defines the project-specific firmware logic, including platform initialization, SimpleSerial setup, command registration, and connection between host-side commands and relevant MAYO functions. The initialization sequence begins with `platform_init()`, `init_uart()`, and `simpleserial_init()`. Commands are registered through `simpleserial_addcmd()`, after which incoming commands are handled in a processing loop based on `simpleserial_get()`[Newj]. Within this structure, the relevant MAYO functions are registered so that they can be invoked from the host computer through ASCII characters defined in `main()`. Listing 3.1 shows the initial contents of the source file. The header files included here provide access to the relevant HAL and SimpleSerial interfaces already available in the ChipWhisperer framework as well as the MAYO interface.

**Listing 3.1:** Initial contents of the source file `simpleserial-mayo1nssam4s.c`.

```
#include <stdint.h>
#include "hal.h"
#include "simpleserial.h"
#include "sam4s.h"
#include "api.h"
#include "mayo.h"

uint8_t test_cmd(...){

    trigger_high();
    for (volatile uint32_t i = 0; i < 200000; i++);
    trigger_low();

    uint8_t ok = 0x99;
    simpleserial_put('r', 1, &ok);

    return SS_ERR_OK;
}

int main(void){

    platform_init();
    init_uart();
    trigger_setup();

    simpleserial_init();

    simpleserial_addcmd('t', 0, test_cmd);

    while (1) {
        simpleserial_get();
    }
}
```

5. **Create and configure the Makefile** The project `Makefile` was also created within the project folder. This file, shown in Listing 3.2, specifies the target build name, the source files to include, the selected SimpleSerial version, the architecture-specific compiler flags, and the inclusion of the required ChipWhisperer build files. In this setup `TARGET = simpleserial-mayo1nssam4s` defines the name of the firmware project, while the source files listed in `SRC` determine which C source files are compiled and included in the final firmware image flashed to the target. The variable `ASRC` serves the same role for assembly source files. SimpleSerial V2 was selected through `SS_VER = SS_VER_2_1`, while the

crypto part of the build system was disabled through `CRYPTO_TARGET = NONE`. The flags `-mcpu=cortex-m4` and `-mthumb`, added through `CFLAGS` and `ASFLAGS`, ensure that the firmware is built for the targets' ARM Cortex-M4 architecture and their Thumb instruction set [GNU; STM; Mic]. Finally, the included build files `../simpleserial/Makefile.simpleserial` and `Makefile.inc` provide the SimpleSerial and main ChipWhisperer build instructions.

**Listing 3.2:** Makefile used to build the MAYO implementation for a Cortex-M4 target.

```
TARGET = simpleserial-mayo1nssam4s

SRC += simpleserial-mayo1nssam4s.c
SRC += api.c
SRC += mayo.c
SRC += params.c
SRC += mem.c
SRC += arithmetic.c
SRC += aes_ctr.c
SRC += aes-publicinputs.c
SRC += fips202.c
SRC += keccakf1600.c
SRC += randombytes.c

ASRC += aes-encrypt.S
ASRC += aes-publicinputs-asm.S
ASRC += aes-keyschedule.S

SS_VER = SS_VER_2_1

CFLAGS  += -mcpu=cortex-m4 -mthumb
ASFLAGS += -mcpu=cortex-m4 -mthumb

FIRMWAREPATH = ..
CRYPTO_TARGET = NONE

include ../simpleserial/Makefile.simpleserial
include ../Makefile.inc
```

6. **Build the firmware for the target platform** Once the `Makefile` and the project source file are in place, the firmware is ready for compilation. For the SAM4S target, this is done using `make` with the platform setting `PLATFORM=CWHUSKY`. The same build process was later used for the STM32F4 target, with the platform setting changed to `PLATFORM=CW308_STM32F4` and with minor adaptations to filenames and include paths. If the build completes successfully, a `.hex` file is generated and can be used for flashing.

**Triggers**
Triggering is used to ensure that trace capture begins at a meaningful point relative to the target computation. The ChipWhisperer Husky supports several trigger methods, including ADC level triggering, UART triggering, Sum of Absolute Differences (SAD) triggering, and edge-count triggering [Newa]. In the experiments conducted in this thesis, GPIO-based rising-edge triggering is used. A GPIO pin is a general-purpose digital input/output pin that can be driven high or low by the firmware. The functions `trigger_high()` and `trigger_low()`, defined in HAL, drive this pin high immediately before the relevant computation and low immediately after it, thereby delimiting the specific code region to be captured. Listing 3.1 shows an example of how these functions are used in the test command.

**Software**

The software layer refers to the host-side environment running on the computer used for experimentation. In this layer, the ChipWhisperer Python interface is used to control the capture device and interact with the target board. A Jupyter Notebook is created as an environment to set up the scope, flash firmware, send SimpleSerial commands, capture traces and perform the analysis. Listing 3.3 shows the commands used to initialize Simpleserial V2 host-side communication, according to the Simpleserial documentation [Newj]. The compiled firmware can be programmed to the target board using ChipWhisperer's target-specific programmer classes: `SAM4SProgrammer` for the SAM4S target and `STM32FProgrammer` for the STM32F4 target. Both methods rely on the microcontroller's built-in bootloader to flash the compiled `.hex` file to the target device.

**Simpleserial Communication**
To ensure successful communication between the host, the ChipWhisperer Husky, and the target device, the same SimpleSerial version must be specified consistently in the `Makefile` (Listing 3.2), the source file (Listing 3.1), and the host-side setup (Listing 3.3). In addition, the communicating devices must share the same UART parameters, including the baud rate, which defines the number of transmitted bits per second [VO22].

Although the 5.7.0 version of the documentation states that SimpleSerial V2 operates at a baud rate of `230400 bps`, it also notes that only the XMEGA and STM32 platforms currently switch automatically to, and can operate at, this higher rate [Newi]. Consequently, for the SAM4S target board, the baud rate must be explicitly set to `38400 bps`, corresponding to the baud rate used by SimpleSerial V1. This is done in Listing 3.3 with `target.ser.baud = 38400`. For the STM32F4 target, no such manual adjustment is required.

To verify the communication setup, the ASCII command `t` is sent to the target. This command is implemented as a test function in `simpleserial-mayo1nssam4s.c`, Listing 3.1. The target is expected to return the value `0x99`, as defined in the source file. If `print(resp)` displays this value, the SimpleSerial communication works as intended.

**Listing 3.3:** Host-side setup code in Jupyter Notebook for SAM4S target using the ChipWhisperer Python library.

```
import chipwhisperer as cw
import time
scope = cw.scope()
scope.default_setup()
target = cw.target(scope, cw.targets.SimpleSerial2)

cw.program_target(
    scope,
    cw.programmers.SAM4SProgrammer,
    "../../firmware/mcu/simpleserial-mayo1nssam4s/simpleserial-
    mayo1nssam4s-CWHUSKY.hex")

def reset_target(scope):
    scope.io.nrst = 'low'
    time.sleep(0.05)
    scope.io.nrst = 'high'
    time.sleep(0.05)

target.ser.baud = 38400

#Simpleserial communication test
target.simpleserial_write('t', b'')
resp = target.simpleserial_read('r', 1)
print(resp)
```

### 3.1.2 Implementations of MAYO

The MAYO signature scheme initially adopted a bitsliced representation, main branch of artifacts url in [BCC+24], for organizing field elements in memory and registers. In the round 2 version of the scheme, this was changed to a nibble-sliced representation, which packs two 4-bit field elements into a single byte, due to significant performance advantages across various platforms [BCC+24].

As the attack reproduced in this thesis targets a nibble-sliced MAYO implementation, the experimental setup is based on the same representation. The implementation used corresponds to commit `29b1436` of `MAYO-M4` main [PQC25]. According to the repository, this code implements the round-2 version of MAYO and is based on the

nibble-sliced Cortex-M4 implementation, although with adapted parameters [PQC25]. MAYO defines several parameter sets corresponding to different security levels. The experiments in this thesis use the MAYO1 parameter set corresponding to security level 1, whose parameters are summarized in Table 3.1. The selected implementation follows the same generic nibble-sliced code path referenced in `SCA-MQDSA` [VGP+26].

**Table 3.1:** Parameter set for MAYO1 corresponding to security level 1 [BCC+25].

| **Parameter** | **Meaning** | **MAYO1** |
|---|---|---|
| $n$ | Number of variables | 86 |
| $m$ | Number of polynomials | 78 |
| $o$ | Dimension of the oil space | 8 |
| $k$ | Whipping parameter | 10 |
| $q$ | Field size | 16 |
| $f(z)$ | Irreducible polynomial of degree $m$ | $f_{78}(z)$ |
| `salt_bytes` | Salt length | 24 |
| `digest_bytes` | Digest length | 32 |
| `pk_seed_bytes` | Public-key seed length | 16 |
| Secret key size | | 24 B |
| Public key size | | 1420 B |
| Signature size | | 454 B |
| Expanded secret key size | | 141 KB |
| Expanded public key size | | 142 KB |

The `MAYO-M4` repository provides both a reference implementation, denoted `ref`, and an optimized implementation, denoted `m4f`. The `m4f` implementation is intended for Cortex-M4 targets with a Floating-Point Unit (FPU), whereas the reference implementation does not rely on such hardware support. The STM32F4 target provides an FPU [Newc], while the SAM4S target does not [Newd]. In the experimental setup, the reference implementation is used in order to maintain compatibility across both targets and with the implementation setting considered in `SCA-MQDSA` [VGP+26].

For validation, a host-side `MAYO-C` [PQC26] implementation is used to verify the results of computations performed on the target boards. This provides a reference for checking that the reduced firmware setup and the target-side operations behave as expected. In the experiments, key generation is performed once using this implementation, after which fixed seeds are reused during repeated signing runs on the target firmware. As a result, the secret oil space and the corresponding public values remain constant across traces.

### 3.1.3 Hardware Constraints for PQC

The mathematical structures required for quantum resistance introduce additional overhead compared to classical schemes such as RSA and ECC in terms of key and signature sizes, which in turn leads to hardware constraints on resource-constrained devices [Scr25]. Although the compressed MAYO1 keys appear relatively small, these values do not fully capture the true memory requirements during execution. The implementation must also handle the expanded secret and public keys, which reach 141 KB and 142 KB, respectively, as shown in Table 3.1. Thus, even though the stored key material is comparatively small, the working memory requirements of the scheme remain substantial.

The SCA to be reproduced targets the function `P1P1t_times_O`, which performs multiplications involving the expanded public key and the secret matrix **O**. As mentioned, the expanded public key produced by MAYO.ExpandPK has a size of 142 KB. As noted in the Hardware section, the SAM4S target provides 128 KB of flash and 64 KB of SRAM. It is therefore not possible to execute the full target function directly on this platform as the target cannot hold the whole expanded key. Several approaches can be considered to address this limitation. One possibility is to stream the required public values from the host to the target during trace capture. In practice, however, this introduced additional noise into the traces, making it more difficult to visually identify the operations needed for subsequent trace processing. Since visibility of these operations is important for analysis, this approach was not adopted.

**Listing 3.4:** Host-side code for preloading P1 entries to the target.

```
def send_uint64_buffer(target, cmd, arr, chunk_bytes=32):
    raw = np.array(arr, dtype=np.uint64).tobytes()
    for i in range(0, len(raw), chunk_bytes):
        chunk = raw[i:i + chunk_bytes]
        target.simpleserial_write(cmd, chunk)
send_uint64_buffer(target, 'c', P1_entries)
```

Instead, for the initial experiments on SAM4S, the computation on the target was divided into smaller operations. The public data required for each operation could then be preloaded and transferred via SimpleSerial, with a maximum packet size of 249 bytes in SimpleSerial V2 [Newj]. In this way, the captured trace does not suffer from additional noise caused by communication during capture. This preloading step is shown for **P1** in Listings 3.4 and 3.5. The host-side code sends the required public values and initialization data in chunks, while the firmware-side code reconstructs these values in local buffers prior to capture.

**Listing 3.5:** Firmware callback used to preload P1 entries from the host.

```
static uint64_t P_buffer[NUM_ENTRIES * LIMBS];
static uint32_t P_offset = 0;

uint8_t cmd_load_Pentries(...){

    memcpy((uint8_t*)P_buffer + P_offset, buf, len);
    P_offset += len;

    if (P_offset >= sizeof(P_buffer))
        P_offset = 0;

    return SS_ERR_OK;
}

int main(void){

    simpleserial_addcmd('c', 0, cmd_load_Pentries);
}
```

## 3.2 Attack Scope

The scope of the attack is to recover secret information from the oil matrix **O** in MAYO by reproducing the SCA described by Vishwaajith et al. in `SCA-MQDSA` [VGP+26]. The attack is a non-profiled, correlation power analysis attack that targets the multiplication technique used in the function `P1P1t_times_O`, simplified in listing 3.6. This function is invoked during the MAYO.Sign algorithm, specifically as part of the expanded secret-key computation in MAYO.ExpandSK.

The ultimate objective of the full attack is recovery of the whole secret oil matrix **O** by sequentially recovering its individual coefficients, thereby compromising the signature scheme. The work in this thesis focuses on experimentally reproducing the attack methodology to validate the leakage assumptions underlying this recovery. While a single-trace setting represents the ideal attack scenario, practical hardware constraints and trace noise do not allow it to be achieved directly in all parts of the attack reproduction.

The SAM4S target board is used for the initial experiments, where the objective is to develop and validate the leakage model in a controlled setting. The resulting method is then carried over to the STM32F4 target board, where it is applied to the full target function and to a trace that require more processing and segmentation.

### 3.2.1 Attack Model

Figure 3.3 outlines the attack workflow considered in this thesis, following the method used in `SCA-MQDSA` [VGP+26]. The workflow consists of six steps:

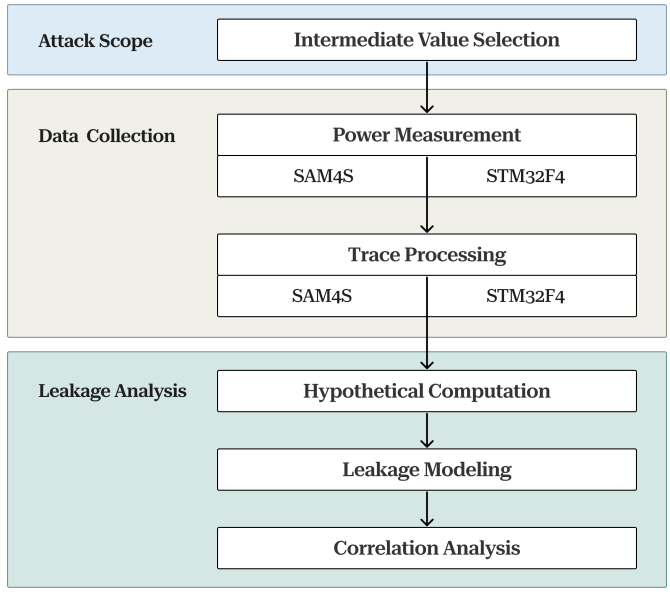



**Figure 3.3:** Overview of the attack workflow following the method described in `SCA-MQDSA` [VGP+26].

- **Intermediate Value Selection** An intermediate value is selected that depends on known public data and secret information.
- **Power Measurement** Power measurements are captured during execution of the target function.
- **Trace Processing** Power traces are segmented and processed in order to isolate the relevant leakage points.
- **Hypothetical Computation** Based on the selected intermediate value, hypothetical internal values are computed for all candidates of an oil position.
- **Leakage Modeling** The hypothetical internal values are mapped to predicted leakage through a leakage model.
- **Correlation Analysis** Correlation analysis is used to determine which candidate hypothesis is most consistent with the measured power.

The intermediate value selection is described within the 3.2 Attack Scope section, while power measurement and trace processing are treated in the 3.3 Data Collection section, where different procedures are required for the SAM4S and STM32F4 platforms. The remaining steps, hypothetical computation, leakage modeling, and correlation analysis are described in the 3.4 Leakage Analysis section.

The attack targets `P1P1t_times_O`, which is a function in the MAYO signing procedure that performs secret-dependent multiplications including the public matrix **P1** and the secret oil matrix **O**. A simplified version of this function is shown in Listing 3.6. The intermediate results of these multiplications are stored in an accumulator, i.e., a buffer that holds the running partial sums and is updated throughout the computation. This function repeatedly invokes `m_vec_mul_add`, which takes as input the number of limbs to process, `m_vec_limbs`, a selected **P1** entry, a secret oil coefficient, and an accumulator position. Internally, `m_vec_mul_add` calls `mul_table` to prepare the values required for the multiplication.

**Listing 3.6:** Simplified structure of the target function `P1P1t_times_O`, where $V = 78$, OIL $= 8$, and `m_vec_limbs`=5 for the parameter set used in the experiments.

```
void P1P1t_times_O(P1, O, acc) {
    entry = 0;
    for (int r = 0; r < V; r++) {
        for (int c = r; c < V; c++) {
            if (c == r) {
                entry++;
                continue;
            }

            P1_entry = P1[entry];
            for (int k = 0; k < OIL; k++) {
                m_vec_mul_add(m_vec_limbs, P1_entry, O[c][k], acc[r][k]);
                m_vec_mul_add(m_vec_limbs, P1_entry, O[r][k], acc[c][k]);
            }

            entry++;}}}
```

Following the implementation of `m_vec_mul_add` and `mul_table` in Listing 3.7, a fixed field element $a$ within $\mathbb{F}_{16}$ is first passed to `mul_table(`$a$`)`, which precomputes the four partial products associated with the four bit positions of a 4-bit field element. This avoids recomputing the same partial products for each multiplication. The routine `m_vec_mul_add` then uses these values to multiply packed $\mathbb{F}_{16}$ elements in parallel. The value $a$ corresponds to a secret oil coefficient, while the input vector **in** is a block of **P1** consisting of `m_vec_limbs` $= 5$ 64-bit limbs. Since one $\mathbb{F}_{16}$ element is represented by 4 bits, each 64-bit limb contains 16 field elements. For each limb, `m_vec_mul_add` extracts the four bit positions across all packed elements, combines the corresponding partial products from `mul_table(a)`, and xor-accumulates the result into *acc*.

The target function iterates over the upper triangular part of **P1**, excluding the diagonal. For each off-diagonal entry $(r, c)$, it performs two calls to `m_vec_mul_add` for every oil matrix column index $k$, one using the oil coefficient $\mathbf{O}[c, k]$ and accumulating

into $acc[r, k]$, and one using the oil coefficient $\mathbf{O}[r, k]$ and accumulating into $acc[c, k]$. With 8 columns in the oil matrix and $\frac{78\times77}{2} = 3003$ **P1** entries in the upper triangle, this yields $3003 \times 2 \times 8 = 48048$ invocations of `m_vec_mul_add`. Since the oil matrix has dimensions $78 \times 8$, each of its 624 coefficients is involved in $\frac{48048}{624} = 77$ calls to this routine during a single execution of the signing algorithm. This repetition is what enables a single-trace attack, since a single execution contains a sufficient number of leakage observations related to the same secret coefficient.

**Listing 3.7:** Simplified implementation of `mul_table` and `m_vec_mul_add` in nibble-sliced MAYO.

```
static inline uint32_t mul_table(uint8_t b) {
    uint32_t x = b * 0x08040201;
    uint32_t mask = 0xf0f0f0f0;
    uint32_t high_half = x & mask;
    return x ^ (high_half >> 4) ^ (high_half >> 3);
}

void m_vec_mul_add(int m_vec_limbs, const uint64_t *in, uint8_t a,
    uint64_t *acc) {
    uint32_t tab = mul_table(a);
    uint64_t mask = 0x1111111111111111;

    for(int i=0; i < m_vec_limbs ;i++){
        acc[i] ^= ( in[i]       & mask) * (tab & 0xff)
                ^ ((in[i] >> 1) & mask) * ((tab >> 8)  & 0xf)
                ^ ((in[i] >> 2) & mask) * ((tab >> 16) & 0xf)
                ^ ((in[i] >> 3) & mask) * ((tab >> 24) & 0xf);
    }
}
```

In order to perform CPA, an intermediate value that provides exploitable leakage must be identified. For CPA to be effective, this value must depend on both known input data and secret information, enabling hypothetical leakage values to be computed and compared with measured power traces. As noted by Vishwaajith et al. in `SCA-MQDSA` [VGP+26], neither the computation in `mul_table` nor the individual multiplications inside the `m_vec_mul_add` internal loop provides a sufficiently strong basis for distinguishing the correct key hypothesis on its own. By contrast, the accumulator combines these contributions within each limb update, producing a stronger leakage signal. The accumulator is therefore selected as the primary target for the analysis. Each invocation of `m_vec_mul_add` performs `m_vec_limbs` $= 5$ accumulator updates, followed by a store operation of the result. Based on the previous computations, this results in $77 \times 5 = 385$ leakage points for each position in the oil matrix $\mathbf{O}$.

### The Accumulator

To use the accumulator as a leakage target, it is necessary to understand both how it is updated and how its state can be reconstructed for each oil position in the oil matrix **O**. Since the accumulator combines contributions from several secret-dependent multiplications, each update must be interpreted in the context of the already accumulated values.

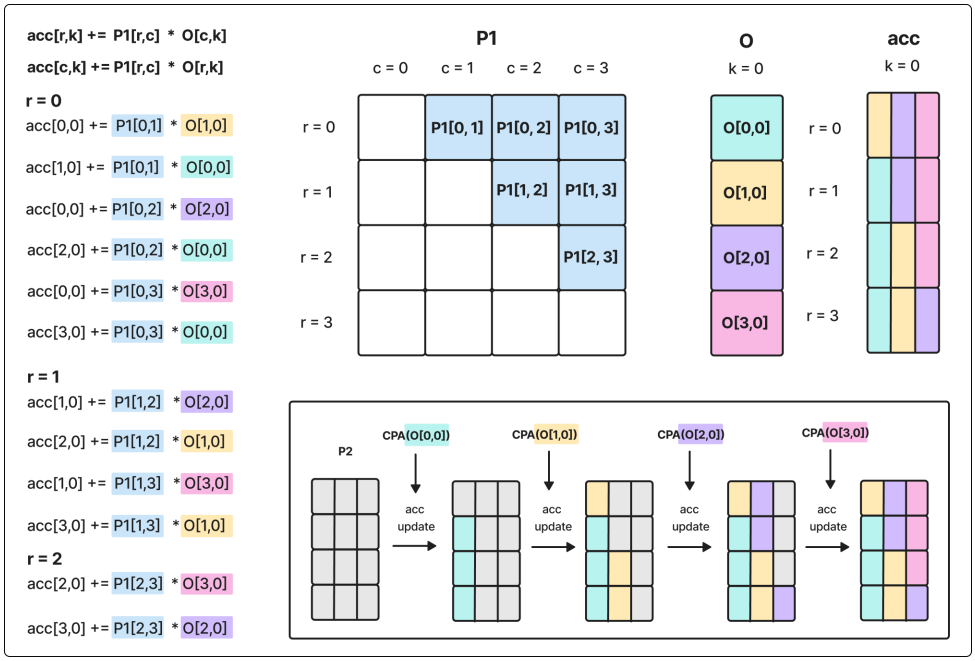



**Figure 3.4:** Simplified explanation of the accumulator update in `P1P1t_times_O`.

To illustrate this update pattern, Figure 3.4 considers a simplified example in which the **P1** and *acc* structures are shown in reduced dimension. For a fixed oil index $k$, each **P1** entry $(r, c)$ contributes to two accumulator positions, namely $acc[r, k]$ and $acc[c, k]$, through multiplication with $\mathbf{O}[c, k]$ and $\mathbf{O}[r, k]$, respectively. This follows directly from `m_vec_mul_add` calls, in which the index $k$ remains unchanged between the oil-matrix access and the corresponding accumulator update. Hence, the accumulator state evolves sequentially within a fixed oil column as the computation proceeds.

Reconstructing this state requires the public entries from **P1** together with the correct initial accumulator state for the coefficient under analysis. This initial value is given by **P2**, since the accumulator is initialized with this value before updates from the oil matrix are applied. Figure 3.4 shows how this makes it possible to perform CPA on the coefficient $\mathbf{O}[0, 0]$, since this is the only position within column

$k = 0$ of the oil matrix whose contributions to the accumulator can be isolated at this point. This is illustrated by the accumulator column in the top-right part of the figure. From left to right, the turquoise contributions corresponding to $\mathbf{O}[0, 0]$ can be reached directly, while yellow, purple and pink contributions remain masked by previously accumulated values. Once the turquoise contribution has been identified and removed, the yellow contribution becomes isolatable, followed in turn by the purple and pink contributions. This means that the value of $\mathbf{O}[0, 0]$ must be recovered correctly in order to maintain the accumulator state required to attack the next coefficient $\mathbf{O}[1, 0]$, as illustrated by the accumulator update sequence in the figure.

Correct reconstruction of this update order is necessary to mirror the firmware computation for each oil coefficient within a fixed oil column. For a given column $k$, the coefficients are recovered sequentially from position 0 to position 77, and each CPA step therefore depends on the correctness of the preceding state updates in that same column. This sequential dependency also provides an internal consistency check, as poor CPA results across several iterations may indicate that an earlier coefficient was guessed incorrectly. Since the first recoverable coefficient is $\mathbf{O}[0, 0]$, the methodology begins by developing and validating the CPA model for this coefficient before extending the same procedure to subsequent coefficients.

### 3.2.2 Simplifications

The full attack targets a trace containing all 48048 calls to `m_vec_mul_add`, which corresponds to a total of 240240 leakage points. Executing this full trace requires the entire **P1** matrix to be available on the target. As discussed in the hardware constraints section, this is not possible on the SAM4S target board.

For this reason, a simplified version of the attack is used for the initial experiments on SAM4S. Instead of executing the full target function, the firmware is restricted to the `m_vec_mul_add` calls that involve a single oil coefficient. With reference to Listing 3.6, this corresponds to fixing $k$ and selecting only those updates in which the coefficient under analysis is $\mathbf{O}[0, k]$, that is, the updates where either $r = 0$ or $c = 0$ within the chosen column. In this way, the resulting trace corresponds to the CPA problem for a single coefficient rather than an interleaved trace that involves the full oil matrix. For one oil coefficient, this gives 77 calls, corresponding to 385 leakage points, and a much shorter trace to analyze. This simplification does not change the underlying attack model, and the same CPA methodology is applied. To confirm the presence of exploitable leakage, initial experiments assume that the initial accumulator state is known for the coefficient under analysis, including positions beyond the first oil-matrix position. This makes it possible to rerun the firmware for different rows of the oil vector, test multiple values while developing the leakage model, and validate the CPA procedure using shorter traces and less data, while

still preserving the full set of leakage points for the selected position. In this way, the simplified SAM4S setup provides a controlled environment for validating the leakage model and the CPA procedure before moving to the full target function on STM32F4.

For a SAM4S setup that involves recovering an entire oil vector, the firmware must be executed repeatedly. After each recovered coefficient, the accumulator state has to be updated with the guessed value and loaded to the target before traces for the next coefficient can be collected.

### 3.2.3 Scalability

Although simplified, the SAM4S setup corresponds to the CPA step used in `SCA-MQDSA`. Starting from $\mathbf{O}[0,0]$ the same coefficient recovery methodology can be applied to subsequent coefficients, provided that the accumulator state is updated correctly after each recovered value. The purpose of the simplified setup is both to identify exploitable leakage points and to demonstrate that CPA can distinguish the correct coefficient candidate. Repeating this process after each iteration of `P1P1t_times_O` extends the attack from a single coefficient to an oil vector, and ultimately to the full oil matrix. In this way, the simplified setup provides a scalable foundation for the complete attack.

While the underlying attack model scales naturally, its practical realization remains target dependent and requires adaptation of the power measurement and trace processing steps.

## 3.3 Data Collection

This section concerns the acquisition of power traces and the processing required before the leakage can be modeled. These steps differ between the SAM4S and STM32F4 platforms, both because the full target function cannot be executed on SAM4S and because the resulting traces differ in length and structure. The purpose of this section is therefore to describe how traces are captured on each platform and how they are prepared for subsequent leakage analysis.

### 3.3.1 Power Measurement

Power measurement refers to the acquisition of raw power traces during the execution of the selected target computation. This is done by arming the scope, triggering the execution of the relevant target-side callback, and recording the resulting signal while the trigger is active. Although the same overall principle of power measurement is used on both platforms, the conditions under which traces are captured differ between them.

### SAM4S

For the SAM4S target board, the power measurement is performed using a firmware callback that isolates the 77 relevant invocations of `m_vec_mul_add` for the oil coefficient under analysis. This simplified setup is shown in Listing 3.8, where this callback is registered using the ASCII command `a`. The trigger functions are placed around the loop so that the captured trace contains only this part of the computation.

**Listing 3.8:** Simplified target function callback for power measurement on SAM4S.

```
uint8_t attack_one_coeff(...){

    trigger_high();
    for (int i = 0; i < NUM_ENTRIES; i++) {
        m_vec_mul_add(
            m_vec_limbs,
            &P_buffer[i * m_vec_limbs],
            target_oil_coeff,
            &acc_work_buffer[i * m_vec_limbs]
        );
    }
    trigger_low();
    return SS_ERR_OK;

int main(void)
{
    simpleserial_addcmd('a', 0, attack_one_coeff);
}
```

A prerequisite for this callback to work as intended is that the required values have already been pre-loaded onto the target, as described in the 3.1.3 Hardware Constraints section. Additional callbacks were also implemented, as can be seen in Table 3.2, to verify that the firmware behavior during execution matches the implementation of the host-side reference implementation `MAYO-C`. More specifically, the command `g` was used to print the accumulator state after the update, allowing it to be compared with the corresponding host-side result computed from the same public and private seeds.

The Jupyter Notebook host-side code for power capture is shown in Listing 3.9. Before capture, the scope is armed and the command `a` is sent to the target. Because the capture is trigger-controlled, it does not begin until `trigger_high()` is reached in the firmware code, unless it never meets the trigger and times out. The capture length is initially set through `scope.adc.samples`. After an initial trace has been recorded, `scope.adc.trig_count` is used to determine how long the trigger signal

remained active, and thereby also how long the target operation is [Newe]. The parameter `scope.adc.samples` is then adjusted accordingly for subsequent captures. This technique also applies to the STM32F4 target board.

**Table 3.2:** SimpleSerial ASCII commands and corresponding firmware callbacks used on SAM4S.

| ASCII | Callback | Purpose |
|---|---|---|
| a | `attack_one_coeff` | Execute target function for oil coefficient |
| c | `load_Pentries` | Load **P1** entries |
| g | `read_acc_out_rows` | Read accumulator output |
| i | `load_acc_init_rows` | Load initial accumulator state |
| o | `set_target_o` | Set target oil coefficient |
| t | `test_cmd` | Test communication |
| z | `reset_acc_out_read` | Reset accumulator read pointer |

**Listing 3.9:** Code for host-side initiated power measurement on SAM4S.

```
target.ser.baud = 38400
scope.adc.samples = 101524
scope.trigger.triggers = 'tio4'
scope.adc.timeout = 3

scope.arm()

target.simpleserial_write('a', b'')
target.simpleserial_wait_ack()

ret = scope.capture()
trace = scope.get_last_trace()
```

### STM32F4

For the STM32F4 target board, the objective is to capture a trace containing the execution of the full `P1P1t_times_O` function so that a single trace includes all 240240 leakage points. This requires a substantially longer capture than in the SAM4S setup and therefore also adapted scope settings.

To determine suitable scope settings, capture errors were used as feedback during the configuration process, guided by the status indicators and diagnostic information provided by the ChipWhisperer Husky. During capture, red blinking LEDs provided an initial indication that the measurement had failed. The cause was then investigated from the host side by inspecting `scope.errors`, `scope.adc.errors`

and `scope.XADC.status`. After correcting the relevant configuration, the command `scope.errors.clear()` was used to return the Husky to a healthy state before starting a new capture attempt.

Using these diagnostics to identify capture errors, together with the ChipWhisperer documentation [Newh] to determine how to resolve them, made it clear that full-function capture required several adjustments to the scope configuration. First, FIFO overflow errors occurred when attempting to capture the full trace at higher sampling settings. This was resolved by reducing the ADC clock rate through `scope.clock.adc_mul = 1`. Second, the ADC timeout had to be increased, since the full target function takes significantly longer to execute than the reduced SAM4S callback. Third, clipping errors indicated that the signal amplitude was too large for the ADC range, which was addressed by lowering `scope.gain.db`. During the process of testing to find a sufficient gain setting, the "gain too low" warning was also useful as an indication that the chosen gain had become too small.

As discussed in the hardware section, the Husky supports streaming mode, which made it possible to capture a trace of this length. The streaming mode was enabled through `scope.adc.stream_mode = True`. Successful use of this mode required the reduced ADC clock setting mentioned above for the FIFO overflow error. Listing 3.10 shows the resulting host-side configuration used to capture the full target function on STM32F4.

**Listing 3.10:** Host-side configurations for capturing the full target function on STM32F4 using streaming mode.

```
scope.adc.stream_mode = True
scope.clock.adc_mul = 1
scope.adc.samples = 14809374
scope.adc.timeout = 10
scope.gain.db = 20

print(scope.errors)
print(scope.adc.errors)
print(scope.XADC.status)
scope.errors.clear()

scope.adc.basic_mode = "rising_edge"
scope.trigger.triggers = 'tio4'

scope.arm()
target.simpleserial_write('a', b'')
ret = scope.capture()
trace = scope.get_last_trace()
```

The callback used to execute the entire `P1P1t_times_0` function on STM32F4 is invoked through the character `a`. All the SimpleSerial ASCII commands for STM32F4 are summarized in Table 3.3. In this setup, the public **P1** matrix is included directly in the firmware as a `.c` file referenced by the `Makefile`, so it is available on the target during execution. The values of **P2** and **O** are still made available through the same pre-loading mechanism used on SAM4S, which explains the role of ASCII commands `i` and `o` in the table.

The resulting raw STM32F4 trace has a length of 14809374 samples. The execution pattern of the target function is not immediately visible in this trace. The expectation was that a repeated pattern in the full raw trace would later support segmentation of the relevant operation. However, this pattern was not sufficiently clear in a single raw trace on STM32F4 due to noise. To make the repeated structure visible, the full target function was therefore captured 100 times and averaged. The averaged trace revealed an underlying repeating pattern that was not clearly visible in a single trace, as can be seen in Figure 3.5. This pattern resembles the repeating behavior reported by Vishwaajith et al. in `SCA-MQDSA` [VGP+26], and provides an important basis for facilitating trace segmentation.

**Table 3.3:** SimpleSerial ASCII commands and corresponding firmware callbacks used on STM32F4.

| ASCII | Callback | Purpose |
|---|---|---|
| a | `run_full_p1p1t_times_0` | Run full `P1P1t_times_0` |
| d | `run_one_rc` | Run 16 `m_vec_mul_add` calls |
| g | `read_full_acc` | Read accumulator |
| i | `load_P2` | Load **P2** into accumulator |
| o | `load_0` | Load **O** |
| t | `test_cmd` | Test Communication |
| y | `run_two_muladd` | Run two `m_vec_mul_add` calls |

### 3.3.2 Trace Processing

Trace processing consists of segmenting a single trace into `m_vec_mul_add` calls and isolating the relevant leakage points. The relevant leakage points must be identified before the leakage model described in the next section can be compared with the captured power traces. The accumulator, and more specifically the store operation following each limb update, is chosen as the intermediate value to be located and modeled. Since memory transfers typically produce clear and identifiable power variations [BCO04], these store operations can serve as useful markers within the trace. Each store operation is preceded by a known sequence of computations, consisting of the `mul_table` calculation, individual multiplications, and the accumulator update.

Knowledge of this algorithmic flow therefore makes it possible to interpret the trace structure and identify where specific activities occur within the trace.

### SAM4S

For the SAM4S target, trace processing began with visual inspection of the raw trace to identify the repeated execution pattern. In contrast to the STM32F4 case, this structure was sufficiently distinguishable to guide the initial segmentation. The objective was to determine which parts of the trace corresponded to one call of `m_vec_mul_add` and how each call is divided into five limb updates.

The full power trace has a length of 101524 samples and contains 77 calls to `m_vec_mul_add`. This gives an expected average call length of approximately 1318 samples. Based on this and visual inspection of the repeated trace structure, the call length and offset were defined. Using these parameters, the trace was segmented into 77 call regions. The resulting segments were then placed on top of each other and aligned.

The next step was to identify the leakage points in each segment. Since the accumulator store operation at the end of each limb was expected to produce the clearest leakage, the last prominent power spike of each limb was initially chosen as the Point of Interest (PoI) for testing. In the first experiments, only two limbs were used to validate the leakage model and to test whether the hypothetical values were modeled correctly. Restricting the initial analysis to two limbs made it possible to isolate the leakage more effectively before extending the model to a larger number of leakage points. Since an incorrect sample position in a single limb can influence the overall result, this simpler setup also made the fine-tuning process more manageable. Furthermore, Vishwaajith et al. report stable correlation within the first 25 sub-traces [VGP+26], corresponding to 125 leakage points, which is covered by the initial $77 \times 2 = 154$ leakage points used here. These tests suggested that the most informative leakage points were located near the end of each limb, but that their exact location drifted slightly across segments. For this reason, when power extraction was extended to all five limbs, it was no longer based on a single fixed sample per limb. Instead, a small sample window was selected near the end of each limb and correlation scans within these windows were used to determine which sample position gave the strongest correlation and was to be used for CPA. The window locations themselves were determined through manual testing and observation of which regions repeatedly gave strong correlations for the correct coefficient.

This window-based extraction method assumes that the correct coefficient candidate also gives the strongest correlation within the selected window of each limb. During the initial experiments, this assumption proved sufficient in many cases and allowed the leakage model and hypothetical computations described in the next section to

be verified. However, the method did not remain robust when extended to longer recovery sequences.

One important factor behind these difficulties is likely the alignment of segments. In order for a specific sample point to correspond to the same operation across segments, the segmentation and alignment must be sufficiently accurate. On SAM4S, this was likely not accurate enough for fixed sample positions to work reliably across all segments. A more robust solution was therefore introduced in the form of a preference rule. The same window correlation scan was still used, but if the same coefficient candidate produced the strongest correlation in three or more limbs within a segment, samples associated with that candidate were favored in the final selection for the remaining limbs. This reduced the influence of strong random correlations and made sample selection more consistent. Although this was not an ideal method for power extraction, it allowed for developing and testing the CPA procedure.

Thus, the initial SAM4S experiments showed that alignment is critical when extracting power from multiple limbs across many segments. In retrospect, this also suggests that the choice of call length and offset should ideally be supported by a more systematic segmentation method than visual inspection and manual tuning alone.

**STM32F4**

For STM32F4, trace processing is primarily concerned with segmenting the complete trace of the target function into individual `m_vec_mul_add` calls and identifying the segments that correspond to a specific oil coefficient. The purpose of this is to isolate the 385 leakage points associated with a single coefficient for use in the later leakage analysis. Since the full execution of `P1P1t_times_O` contains 48048 invocations of `m_vec_mul_add`, this requires the trace to be correctly segmented into 48048 individual call regions, each containing 5 leakage points.

Compared to the SAM4S setup, this segmentation task is considerably more challenging. The STM32F4 traces are longer and noisier, making the repeated call structure difficult to identify directly in a single raw trace. As described in the previous section, an average over 100 full-function captures were therefore used as a starting point in order to reveal the operation pattern hidden beneath the noise. In addition, due to the experiences from the initial experiments on SAM4S, a more systematic segmentation method based on correlation scanning and peak detection is used for STM32F4.

A first estimate of the call length can be obtained from the total call length by dividing the 14809374 samples by 48048 calls, which gives an average call length of approximately 308 samples. This provides a useful starting point when zooming into

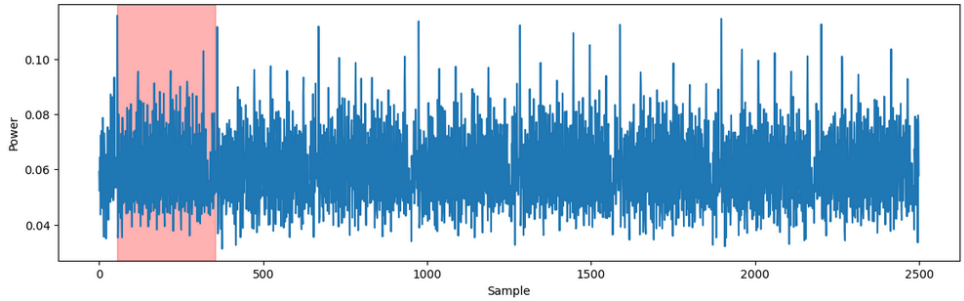



**Figure 3.5:** Average trace revealing the underlying call pattern. The red region marks the template corresponding to a single call.

the averaged trace in Figure 3.5. In the first few thousand samples, a repeated local pattern becomes visible. Guided by the expected call length, one such pattern can be identified as a candidate instance of a single `m_vec_mul_add` call, as can be seen from the area marked in red in the figure. To support this assumption, the auxiliary SimpleSerial commands `y` and `d` were used to capture reduced traces containing two and sixteen `m_vec_mul_add` calls, respectively. These reduced traces made it easier to recognize the local structure of the repeated pattern and thereby supported the assumption that the selected template corresponds to a single call.

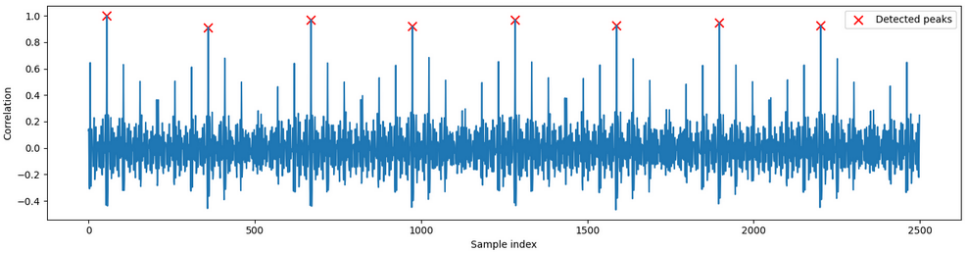



**Figure 3.6:** Resulting correlation trace with identified peaks to be used for segmentation of the full trace.

Once this template had been identified, it was used in a correlation scan across the entire trace to locate similar call patterns. This revealed regular correlation spikes, showing that the same structure is present throughout the trace despite the noise. Peak detection was then applied to the correlation output using `find_peaks` from SciPy's `scipy.signal` module, with adjusted thresholds for peak height and a minimum peak distance of 300 samples, as can be seen in Figure 3.6. Applying this procedure to the raw trace produced the same results as when applying it to the average trace. However, because the average trace is more reliable due to the absence of noise, those results were used for the segmentation and saved for later use.

Examining the distances between successive peaks revealed a regular call-length pattern. For most calls, the distance alternates between 305 and 309 samples. For every sixteenth call, however, the distance increases to 328 samples. This is consistent with the structure of `P1P1t_times_0`, where each **P1** entry results in 16 calls to `m_vec_mul_add`, two calls for each of the eight values of $k$. The increased distance therefore corresponds to the transition to the next column within the same **P1** row. A second larger increase occurs when the row index $r$ changes, in which case the distance becomes 348 samples. Counting the number of occurrences of these distances yielded 24024 calls of length 305, 21021 of length 309, 2926 of length 328, and 77 of length 348. Summarized, this counts to 48048 calls, which supports the interpretation of the observed segmentation pattern.

Once this call structure is known, the full trace can be split into 48048 segments, each corresponding to one call to `m_vec_mul_add`. Since each such call processes five limbs, each segment contains five accumulator updates. When the capture settings remain unchanged, the same segmentation pattern can be used for traces of the same target function collected with different input data. The peak locations can therefore be determined once, for example from the averaged trace, and then used for other traces captured under the same conditions.

After segmentation, the loop logic of the target function can be used to determine which of the 48048 segments correspond to a given oil coefficient. In this way, the 77 segments relevant to a single coefficient can be extracted from the trace. Since these segments have slightly different lengths, it is useful to bring them to a common size before isolation of the leakage points. This is done by cropping the longer segments to a common length of 305 samples, which makes it easier to compare and identify leakage points consistently across all segments.

Compared to the SAM4S case, the identification of PoIs was more straightforward on STM32F4 once the full trace had been segmented. This was mainly due to the power structure of the segments and more precise segmentation. The storage of the accumulator, which is the operation to be isolated, is expected to produce a prominent power spike at the end of each limb, as it is the final operation before the processing of the next iteration begins. Figure 3.7 shows peak detection on the average of the 77 segments corresponding to the oil position $\mathbf{O}[0, 0]$. This yields five peaks at samples 63, 113, 163, 213 and 263, with constant spacing of 50 samples between them. These positions were assumed to correspond to the store operation, as they appear at the end of each limb and show a noticeably higher power consumption than the other operations. Using the leakage model developed on SAM4S, all samples were tested to determine which positions produced the strongest correlations within each limb. These five sample positions gave the best correlations across the limbs and were therefore selected as fixed leakage points for subsequent power extraction.

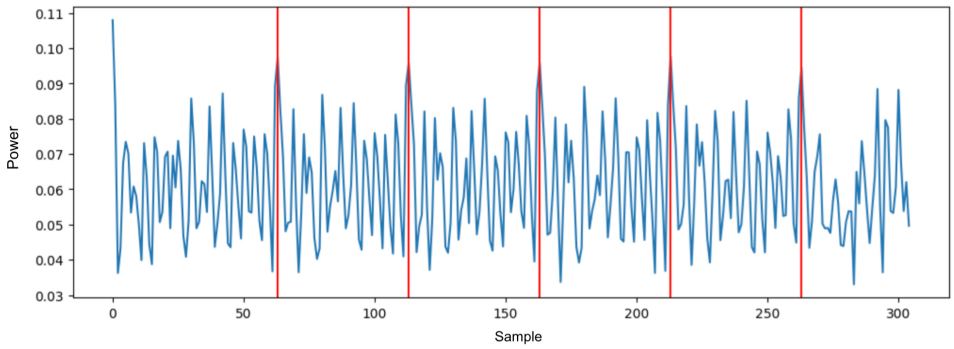



**Figure 3.7:** Peak detection on the average of the 77 segments corresponding to $\mathbf{O}[0, 0]$. The five detected peaks occur at samples 63, 113, 163, 213, and 263.

*What is the minimum number of captures required to obtain an averaged trace that enables correct segmentation?* To answer this question, we performed a series of experiments in which the number of captures included in the averaged trace was gradually reduced. For each averaged trace, peak detection was applied to determine whether all 48048 expected peaks could still be identified. The expectation was that using fewer captures would reduce the correlation strength and make the repeated structure less distinguishable. It was therefore necessary to visually inspect the correlation output and assess the peak amplitudes in order to adjust the minimum peak height used during peak detection. If the expected number of peaks could not be detected after several adjustments of the peak detection parameters, the averaged trace was considered too noisy to enable correct segmentation.

## 3.4 Leakage Analysis

After data collection and trace processing, we have everything we need to proceed with leakage analysis. The purpose of the leakage analysis is to determine which key candidate best explains the observed power consumption. The methodology here was first developed on the SAM4S target board and then transferred to STM32F4. Since the full target function is analyzed on STM32F4, the description in this section uses that setting as the main reference.

### 3.4.1 Hypothetical Computation

The hypothetical intermediate computation step is concerned with reproducing the accumulator updates performed on the target for all possible candidates of the secret oil coefficient, which we will name the hypothesis. Since MAYO uses a nibble-sliced representation over $\mathbb{F}_{16}$, each oil coefficient can take one of 16 possible values.

To mirror the firmware computation, the same operations must be performed in the same order as on the target. This requires both the public values used in the current stage and the corresponding accumulator state. More precisely, the hypothetical computation takes as input the 77 relevant **P1** entries for the coefficient under analysis, together with the associated initial accumulator positions to be updated by the coefficient. For a fixed oil coefficient, these **P1** entries are determined directly by the loop structure of `P1P1t_times_O`. Each relevant trace segment corresponds to one call to `m_vec_mul_add`, and therefore to one specific **P1** entry. As a result, there is a one-to-one relationship between the 77 trace segments associated with a given coefficient and the 77 **P1** entries used to build the hypothesis. Each such **P1** entry consists of five limbs, which matches the five accumulator updates for the corresponding power segment.

**Listing 3.11:** Function for building the hypothesis for each key candidate.

```
def build_hypothesis_hw(P1_entries, acc_init, limbs=5):
    N = len(P1_entries)
    hyp = []

    for key_candidate in F16:
        candidate_hyp = []

        # Compute the intermediate values for this candidate
        for i in range(N):
            acc_after = m_vec_mul_add_model(
                P1_entries[i], key_candidate, acc_init[i])

            # Use the hamming weight as leakage prediction
            for limb in range(limbs):
                acc_limb_32 = int(acc_after[limb]) & 0xffffffff
                candidate_hyp.append(hw(acc_limb_32))

        hyp.append(candidate_hyp)
    return hyp
```

The creation of the hypothesis is implemented by the `build_hypothesis_hw` function shown in Listing 3.11. For each of the 16 candidate values in $\mathbb{F}_{16}$, the function applies a software version of `m_vec_mul_add` and `mul_table` in order to compute the updated accumulator state. Since each call updates five limbs, this yields five intermediate values per iteration. In total, the function therefore produces 385 hypothetical values for each of the 16 key candidates. These values are obtained by mapping the updated accumulator limbs to their Hamming weights, as will be explained in more detail in the next section, 3.4.2 Leakage model. The result is stored in the two-dimensional array `hyp`, which may be viewed as a hypothesis matrix. Each row corresponds to

one candidate value of the secret oil coefficient, while each column represents the predicted HW for a single update, corresponding to one limb in one segment.

Although each accumulator limb is represented as a 64-bit value, Vishwaajith et al. note in `SCA-MQDSA` that the Cortex-M4 hardware operates on 32-bit registers [VGP+26]. Since the same applies for the underlying hardware in the present setup, the hypothetical leakage is computed from the lower 32 bits of each accumulator limb. In `build_hypothesis_hw` this is done by applying the bitwise mask `acc_after[limb] & 0xffffffff` before computing the HW.

### 3.4.2 Leakage Model

The Hamming Weight (HW) leakage model introduced in Section 2.3.1 is used for the leakage analysis in this attack. From the hypothesis calculated in the previous section, we have that for each hypothetical key candidate, the predicted leakage of the updated accumulator value is found by computing its HW. This yields a hypothetical leakage pattern that can be compared with the measured power samples extracted from the corresponding trace segments.

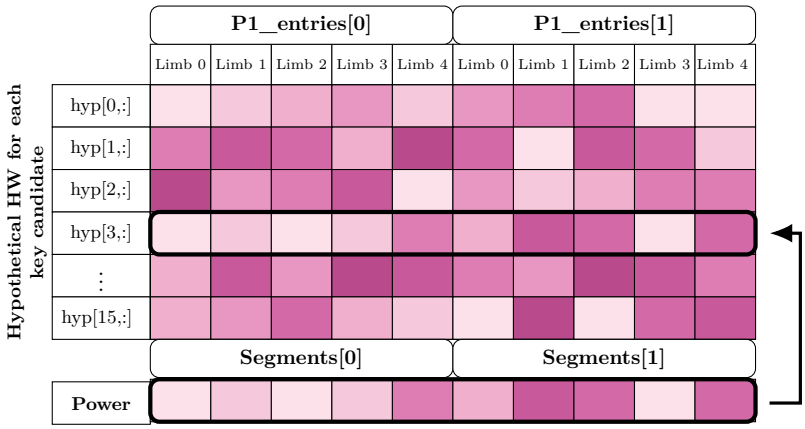



**Figure 3.8:** Visual representation of hypothesis comparison with measured power.

Figure 3.8 illustrates the main idea behind this comparison for the two first segments and **P1** entries. The upper part of the figure represents the hypothesis matrix, where each row corresponds to one of the 16 candidate values for the oil coefficient, and one column corresponds to one leakage point. The lower part represents the measured power values extracted from the trace segments. The color intensity may be interpreted as the magnitude of the value, with darker pink indicating higher measured power in the power vector and a larger number of ones in the hypothesis. The figure also illustrates how one **P1** entry corresponds to a single segment, and how each of the five leakage points within that segment has 16 possible hypothesis

candidates. The goal for the subsequent correlation analysis is to identify the candidate whose predicted leakage pattern shows the strongest agreement with the measured power. In this example, candidate 3 is the best match.

The figure is intended as a conceptual illustration. In practice, the measured power does rarely equal the HW exactly, since the traces are affected by noise and other sources of variation. The correct key candidate is therefore not expected to match the measured power perfectly at every point, as shown in the figure. Instead, the leakage model provides a candidate-specific behavior pattern that can be statistically compared with the measured power trace.

### 3.4.3 Correlation Analysis

Once the hypothetical leakage values and measured power samples are available, the next step is to evaluate how well each key candidate explains the observed trace. For this, the Pearson correlation coefficient is used as the statistical distinguisher. For each candidate, the hypothetical leakage vector is correlated with the measured power vector consisting of the selected PoIs from the corresponding trace segments. For the setup on STM32F4, both positive and negative correlations are considered equal indications of correlation. The analysis therefore considers the absolute value of the Pearson correlation score of each candidate to determine which one shows the strongest linear relationship with the measured power.

This step is implemented by the function `run_cpa` in Listing 3.12. The function takes as input the measured power values, the hypothesis, and the number of limbs. First, the extracted power values are normalized per limb. Then, for each key candidate, the corresponding hypothetical leakage values are normalized in the same way. The NumPy function `np.corrcoef()` is then used to compute the Pearson correlation between the measured power and the hypothetical leakage for that candidate. The output is the vector `corr`, which contains one correlation score for each of the 16 key candidates.

The purpose of this step is thus to determine which candidate produces a hypothetical leakage pattern that is most consistent with the measured trace. The candidate with the strongest correlation is taken to be the most likely value of the secret oil coefficient. The reliability of this comparison strongly depends on the number of available leakage points. If only a small number of points is used, a high correlation may occur by coincidence. In contrast, if the same candidate consistently shows strongest agreement with the power across all 385 leakage points associated with one oil coefficient, the resulting correlation is much less likely to be accidental.

**Listing 3.12:** Pearson correlation between measured leakage and each key hypothesis.

```
def run_cpa(power, hyp, limbs):
    power = normalize_limbs(power, limbs)
    corr = np.zeros(hyp.shape[0])

    for k in range(hyp.shape[0]):
        h = normalize_limbs(hyp[k], limbs)
        c = np.corrcoef(power, h)[0, 1]

        if not np.isnan(c):
            corr[k] = c
    return corr
```

### Recover Oil Coefficient

At this stage, all the components required to recover a single oil coefficient are in place. The previous steps are combined in the function `recover_oil` in Listing 3.13, which is used to recover the secret value at a given oil-matrix position specified by a row $z$ and column $k$.

A prerequisite for this function is that the full trace has already been segmented into 48048 parts and stored in `segment_records`. In addition, the full **P1** matrix must be available, and the `acc_state` must be initialized consistently with the current stage of the attack. For the first coefficient of a column, this corresponds to **P2**. The function also takes as input the row and column $(z, k)$ of the oil coefficient under analysis, together with a list of PoIs specifying from which sample positions the power should be extracted in each limb. Finally, the MAYO parameters $v = 78$, $o = 8$ and $limbs = 5$ must be provided. Given these inputs, the recovery of a single oil position consists of the following steps:

1. The 77 trace segments relevant to the oil coefficient at position $\mathbf{O}[z, k]$ are extracted from the full set of segmented calls.
2. Since the extracted segments may have slightly different lengths, they are merged to a common target length of 305 so that they can be analyzed consistently.
3. The corresponding **P1** entries, the relevant accumulator positions and their initial state are collected for the selected coefficient. These are the same values that are used by the target function for the updates associated with the current oil position.

4. A hypothesis is then built for all 16 possible values of the oil coefficient. This is done by mirroring the firmware computations and mapping the resulting accumulator updates to HWs using `build_hypothesis_hw`.

5. Using the chosen PoIs, one power value is extracted from each limb of each segment. These values are combined into a single power vector containing 385 measured leakage points.

6. The function `run_cpa` is then applied to compare the measured power vector with the hypothesis matrix using the Pearson coefficient. For each candidate oil value, this yields a correlation score describing how well the predicted leakage agrees with the measured power.

7. Finally, the oil coefficient is guessed by selecting the candidate with the strongest absolute correlation. The function returns this guess together with the full correlation vector `corr`.

**Listing 3.13:** Simplified function for recovering a single oil coefficient.

```
def recover_oil(segment_records, acc_state, P1, v=78, o=8, limbs=5, z, k,
    poi_list= [63, 113, 163, 213, 263]):
    #segments
    oil_records = get_oil_segment_records(segment_records, z, k, v, o)
    segments = merge_oil_segments(oil_records, target_len=305,)

    #hypothesis
    p1, acc_init = get_entries_for_coeff(acc_state, P1, v, o, limbs, z, k)
    hyp = build_hypothesis_hw(p1, acc_init, limbs=limbs)

    #power
    power_parts = []
    poi_lists = []
    for i in range(limbs):
        poi = int(poi_list[i])
        poi_lists.append(np.array([poi], dtype=np.int64))
        p = segments[:, poi]
        p = (p - np.mean(p)) / (np.std(p) + 1e-12)
        power_parts.append(p)
    power = np.empty(limbs * 77)
    for i in range(limbs):
        power[i::limbs] = power_parts[i]

    #correlation
    corr = run_cpa(power, hyp, limbs)
    guess = int(np.argmax(np.abs(corr)))
    return guess, corr
```

In this way, `recover_oil` combines trace segmentation, hypothesis construction, leakage modeling and correlation analysis into one procedure for guessing the value of a single coefficient of the oil matrix **O**.

### Recover Oil Vector

The `recover_oil` procedure described in the previous section can be applied iteratively in order to recover an entire oil vector. One oil vector corresponds to one column of the oil matrix and therefore consists of $v = 78$ oil positions. Since the full oil matrix contains $o = 8$ columns, recovery of one oil vector constitutes one step toward recovery of the complete matrix.

Recovery of an oil vector is performed by applying `recover_oil` successively to all 78 row positions of a fixed column. However, this requires the accumulator state to be updated correctly throughout the process. For the first coefficient in the vector, row 0, the accumulator must be initialized from **P2**. Thereafter, after each recovered coefficient, the accumulator state must be updated with the guessed value before the next coefficient can be analyzed. This is done by simulating the corresponding accumulator update in software using one call to `m_vec_mul_add` with the guessed coefficient. In this way, the accumulator state evolves in the same order as in the target computation and can be used as input to the next recovery step.

A central challenge here is that recovery of each coefficient depends on the correctness of the previously recovered ones. If an incorrect value is accepted at one stage, the accumulator state used in later stages will also become incorrect, which may propagate the error further along the vector. For this reason, the method must include a criterion for deciding when a recovered coefficient is sufficiently reliable to be accepted directly and when additional checks are needed. This is handled through an ambiguity margin which measures how much stronger the best correlation is than the second best correlation. If this margin is sufficiently large, the best candidate is accepted directly. If the margin is too small, the result is treated as ambiguous, and a lookahead step is performed, inspired by the approach in `SCA-MQDSA` [VGP+26].

The recovery of an oil vector uses the same prerequisites as the recovery of a single oil coefficient, namely segmented traces, full **P1** matrix, the current accumulator state and the selected PoIs. In addition to these inputs, two further parameters are required. The first is, as mentioned, the `ambiguity_margin`, which defines how much better the winning candidate must be than the next best candidate in order to be accepted directly. The second is `lookahead_top_k`, which determines how many of the strongest current candidates should be considered in the lookahead step.

The recovery procedure begins by analyzing the current row position in the oil vector, using `analyze_row`. This step calls the `recover_oil` function and uses the returned

correlation scores for each candidate to determine the best candidate, its correlation, the next best candidate, the next best candidate correlation, and the margin between them. If the margin exceeds the `ambiguity_margin` threshold, the best candidate is selected as the recovered coefficient. If the margin is smaller than the threshold, the lookahead mechanism is activated.

In the lookahead step, the `lookahead_top_k` strongest candidates for the current row are considered in turn. For each of these candidates, a temporary accumulator state is created by updating the accumulator with the current candidate value. Using this temporary state, the next row in the oil vector is analyzed, and a combined score is then computed from four quantities: the correlation of the current candidate, the margin at the current row, the winning correlation at the next row, and the margin at the next row. This produces a score that reflects both how plausible the current candidate is and how consistent it makes the next recovery step. The candidate with the best total score is then selected as the final guess for the current row.

Once the final guess has been determined, the accumulator state is updated with that value and the recovery proceeds to the next row of the oil vector. Repeating this process for all 78 rows yields one complete vector. The function `recover_oil_vector`, which implements this procedure, is included in Appendix A and shown in Listing A.3.

### Lookahead Example

Table 3.4 summarizes the oil vector recovery results for rows 0 and 1. In this example, the `ambiguity_margin` is set to 0.2, meaning that the best candidate must exceed the second best candidate by at least 0.2 in absolute correlation in order to be accepted directly. For row 0, this condition is satisfied, and the oil coefficient is therefore selected without lookahead. For row 1, however, the result is ambiguous. The absolute correlation of the best candidate is low, and the margin to the next best candidate is only 0.030064, which is below the ambiguity threshold. As a result, the lookahead mechanism is activated. In this case, `lookahead_top_k = 3`, meaning that the three best candidates, ranked by absolute correlation, are evaluated further.

**Table 3.4:** Summary of oil recovery for rows 0 and 1 during oil vector recovery.

| Row | Local best | \|Local corr\| | \|Next-best corr\| | Margin | Decision |
|---|---|---|---|---|---|
| 0 | 8 | 0.380028 | 0.085679 | 0.294349 | Direct |
| 1 | 5 | 0.139192 | 0.109128 | 0.030064 | Lookahead |

Table 3.5 shows the lookahead evaluation for row 1. For each candidate, the current candidate is temporarily assumed to be correct, the accumulator state is updated accordingly, and then the next row is analyzed. The absolute correlation and margin

of the winning candidate in the next row are then combined with the current row statistics to compute a total score. As shown in the table, candidate 5 stands out clearly from the alternatives, with a much stronger next-step result and the highest total score, making it possible to identify 5 as the correct coefficient for row 1 with greater confidence.

**Table 3.5:** Lookahead evaluation for row 1.

| Cand. | \|Curr. corr\| | Curr. margin | Next guess | \|Next corr\| | Next margin | Total score |
|---|---|---|---|---|---|---|
| **5** | **0.139192** | **0.030064** | **0** | **0.378018** | **0.238842** | **0.786115** |
| 9 | -0.109128 | -0.030064 | 6 | 0.082895 | 0.003947 | 0.165906 |
| 12 | 0.102537 | -0.036655 | 10 | 0.119494 | 0.042845 | 0.228221 |

Thus, the purpose of the lookahead mechanism is to answer the question: *what does the recovery result look like at the next step if this candidate is assumed to be correct at the current step?* This makes it possible to verify uncertain decisions before proceeding. Testing indicates that a single lookahead step is sufficient under the current conditions, although extending the method to more lookahead steps remains a possible option for increased robustness.

#### Recover Oil Matrix

For recovery of the full oil matrix, `recover_oil_vector` is applied independently to each of the 8 columns. As shown in function `recover_oil_matrix`, available in Appendix B, recovery proceeds column by column while continuously updating the accumulator state. For each column, the recovered oil vector is inserted into the corresponding column of `recovered_O_matrix`. When all columns have been processed independently, the function returns the complete guessed oil matrix together with the final accumulator state. The recovered accumulator state is compared with the accumulator state produced by the `MAYO-C` implementation using the same input values, and the accumulator state produced by the firmware in order to verify the correctness of the computations.

### 3.4.4 Remarks

The methodology described here was applied to several datasets by saving the values of **P1**, **P2** and **O** from the executions of the host-side `MAYO-C` implementation and using them both on the target and in the Jupyter Notebook environment. When the capture settings remained unchanged, the same segmentation pattern could be reused across datasets. This made it possible to test the recovery procedure and the lookahead parameters on more than one dataset. In addition, the methodology was applied to the average trace as well as the raw trace for comparison.

For the SAM4S setup, one possible improvement would be to replace the visually guided segmentation procedure with a more systematic correlation-based approach similar to the template-based method used on STM32F4. Such a method would likely have simplified the segmentation stage and made the subsequent power extraction more robust and accurate across traces.

The methodology is structured so that the identification of leakage points is a part of trace processing. However, a methodological observation is that the identification of leakage points was not a strictly separate step from the later leakage analysis. In practice, the development of the hypothetical computation, the leakage model, and the selection of leakage points were strongly interdependent. This was especially the case for the SAM4S setup, where verifying that the hypothetical values were modeled correctly required sufficiently accurate leakage point selection, while the identification of suitable leakage points in turn depended on having a reasonably accurate model. As a result, the methodology was developed through an iterative process of testing, refinement, and correction.

## 3.5 Countermeasure Evaluation

Once the attack setup, measurement procedure, and leakage analysis method have been established, the same methodology can be extended to assess the effectiveness of countermeasures under comparable conditions. The countermeasure evaluation considers masking with both two and three shares.

We evaluate masking as a countermeasure against the side-channel attack described in the previous sections. The masking approach for the target function `P1P1t_times_O` is inspired by the masked version of UOV [KNK+25], another multivariate candidate in the NIST additional signatures process. In the masked version of UOV, mUOV, masking is introduced as a countermeasure to protect sensitive components by representing sensitive values using multiple random shares instead of processing them directly. Since both schemes are multivariate-based, they rely on similar arithmetic operations. This also applies to the secret-dependent operation `P1P1t_times_O`, which is part of the same secret key expansion step in mUOV as in MAYO. The countermeasure evaluation is limited to the target function, meaning that only the required input for this function is generated. Therefore, interoperability with the preceding and following operations is not implemented. Instead, correctness of the masked computation is verified by comparing the reconstructed output with the corresponding `MAYO-C` implementation on the host computer.

### 3.5.1 Masked Representation of O

The masking was implemented by representing each secret oil value of **O** as the XOR-combination of multiple shares. For a coefficient $\mathbf{O}[z, k]$, the masked representation is defined as

$$O[z, k] = O_0[z, k] \oplus O_1[z, k] \oplus \cdots \oplus O_{n-1}[z, k],$$

where $n$ is the number of shares. Random values are generated for all shares except share zero. Share zero is then computed by XORing the original value with the randomly generated shares, ensuring that the XOR of all shares reconstructs the original value. The sharing procedure is performed before the targeted computation is executed. During the masked execution, the computation is performed separately on each share instead of directly on the oil values.

The randomness used for generating the random shares is obtained from the Random Number Generator (RNG) available on the STM32F4 platform through the randombytes files. For this to compile correctly, `randombytes.c` should include the correct STM32F4 HAL RNG header, located in the ChipWhisperer firmware environment as `hal/chipwhisperer-fw-extra-stm32f4/stm32f4xx_hal_rng.h`.

### 3.5.2 Experimental Setup

The masked implementation was evaluated using the STM32F4 target. A new firmware project was created by copying the contents of `simpleserial-mayo1nsstm` into a new directory named `simpleserial-mayo1masked`. The firmware implementation file was renamed to `simpleserial-mayo1masked.c` and modified for the masked experiment. The values of **O** were replaced by their shared representation, and the computation was adapted to process the shares separately.

The power measurement procedure follows the same setup as for the unmasked STM32F4 experiments. The only difference is that the masked firmware is compiled, and the resulting `.hex` file for the masked implementation is flashed onto the target board instead.

For the two-share implementation, fresh shares are generated for each masked `P1P1t_times_O` computation using the RNG on the STM32F4 target. This means that the masked values differ between captures, and the CPA attack is therefore performed on raw traces rather than on averaged traces. For three-share implementation, we introduced fixed shares to make it possible to average several captures and obtain clearer leakage profiles to easier evaluate the share-wise leakage.

### 3.5.3 Trace Processing

Splitting each sensitive value into multiple shares changes the resulting power trace. In the masked implementation, each share must be processed separately, meaning that the number of calls to `m_vec_mul_add` increases linearly with the number of shares. Since the unmasked implementation performs 48048 calls, the two-share implementation performs $48048 \times 2$ calls, while the three-share implementation performs $48048 \times 3$ calls. The same approach as described in Section 3.3.2 Trace processing for STM32F4 is applied. Since the masked implementation processes shares separately, the peak detection must identify one segment for each share-dependent call in order to ensure that all relevant operations are included. The logical structure of the masked target function is then used to collect and group the segments that operate on the same share.

After segmentation, peak detection is applied to an average computed over 77 segments for a single share in order to locate the five power-expensive store operations. These points are then used as PoIs for the CPA attack.

### 3.5.4 CPA on Masked Shares

The attack procedure to recover one secret oil value, described in Section 3.4, is applied separately to each share. This means that the same leakage model and correlation-based analysis are applied to each share, producing one recovered value per share. After all shares have been analyzed, the recovered share values are combined using XOR, which gives the final guess for the oil position under analysis.

The oil-vector and oil-matrix recovery methodology is also applied to the masked implementation. The lookahead mechanism and overall recovery procedure remain unchanged, but recovery is performed independently for each share. Thus, for each oil vector, two share vectors must be recovered and XORed together in the two-share implementation, while three share vectors must be recovered and XORed together in the three-share implementation. The recovered values are compared with the true oil matrix for the current dataset to determine whether the attack is successful against masked implementations as well.

# Chapter 4
# Results and Analysis

In this chapter, we present the results obtained from the methodology. We divide the chapter into four main parts: 4.1 Data Collection, 4.2 CPA Attack, 4.3 Countermeasure Evaluation, and finally 4.4 Summary of Results. In each part, the experimental results are presented and analyzed together, allowing the observations and their implications to be discussed in context.

In accordance with open-science principles, the firmware and attack notebooks developed in this thesis are made available in a public GitHub repository[1]. The repository includes selected trace data that can be used to inspect and rerun parts of the unmasked and two-share masked attacks discussed in this chapter. It also provides firmware and a capture notebook to acquire power traces for the remaining experiments. Larger datasets, such as averaged traces and the three-share masked trace, are omitted due to GitHub file-size limitations. Details on the required data files, hardware setup, and directory structure are provided in the repository `README`.

## 4.1 Data Collection

In this Section we present the results from the power measurement and trace processing stages of the methodology. We first show three power traces to illustrate the measured power characteristics of the different target boards and averaged captures. We then show the results of applying the correlation-based segmentation technique described in Section 3.3.2 for the full target function on the STM32F4 target board.

### 4.1.1 Power Measurement

For the power measurement we have three cases: the raw capture on SAM4S target board, raw capture on STM32F4 target board and average capture on STM32F4 target board.

[1] https://github.com/noramvi/sca-mayo

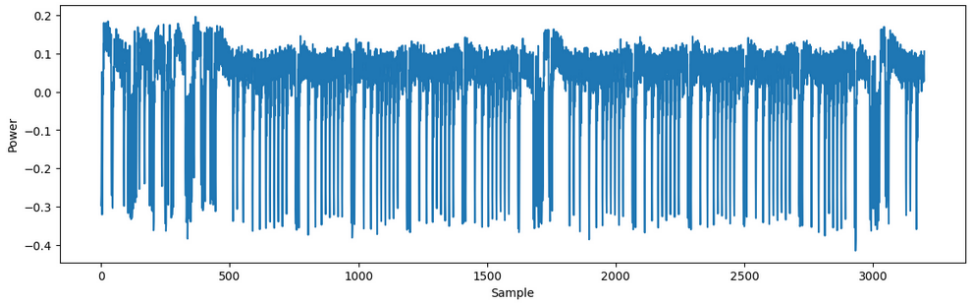



**Figure 4.1:** Raw power trace SAM4S.

The output of the measurement procedure on SAM4S is a power trace consisting of 101524 samples. The entire trace corresponds to 77 `m_vec_mul_add` operations on a single coefficient, which makes a controlled power trace suitable for initial analysis. Figure 4.1 shows the first 3000 samples of the raw power measurement, where the power displayed in the figure corresponds to two `m_vec_mul_add` operations on a single oil coefficient. A pattern can be observed repeating five times before a short pause, after which the same pattern repeats again. This corresponds well with one call to `m_vec_mul_add`, which performs five limb operations for each function call.

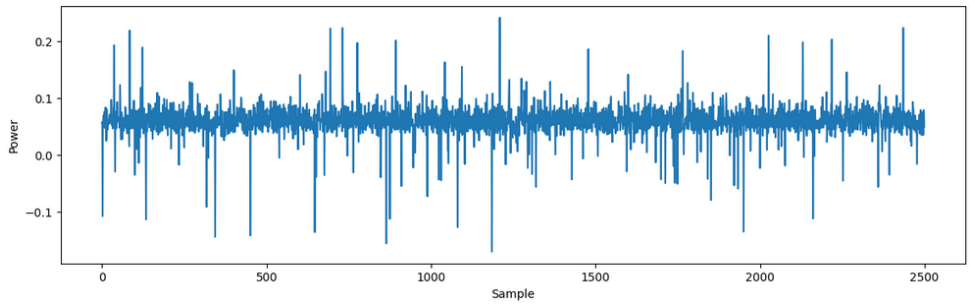



**Figure 4.2:** Raw power trace STM32F4.

The first 2500 samples of the resulting power trace from the STM32F4 target board are shown in Figure 4.2. As described in the methodology, the repeated operation pattern is not directly visible in the raw trace. Instead, the trace contains large power spikes with varying distances between them. Figure 4.3 shows the result of averaging 100 captures. In this trace, a consistent repeated pattern becomes visible. This indicates that, when capturing the full target function, the operation pattern is present in the measured signal but is partially hidden by noise in a single raw capture.

Comparing the pattern visible in the averaged STM32F4 trace with the pattern observed on SAM4S shows that the call iterations are represented by more samples

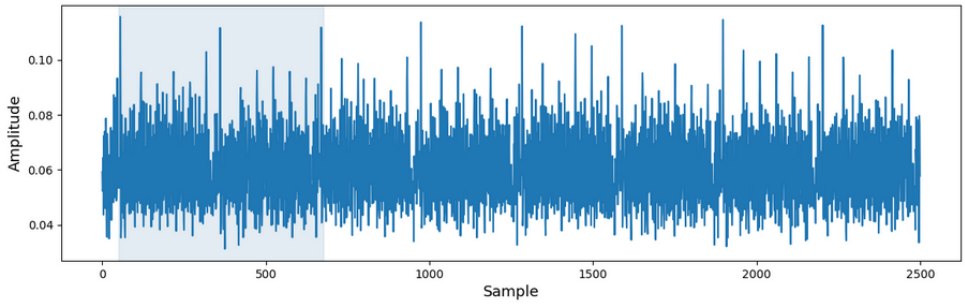



**Figure 4.3:** Mean power trace of 100 captures on STM32F4, with two `m_vec_mul_add` calls marked in blue.

and are more clearly separated in the SAM4S trace, since SAM4S was sampled at a higher rate. Figure 4.4 shows a zoomed-in view of the highlighted region from the STM32F4 trace in Figure 4.3, corresponding to two calls. The repeated pattern is more clearly visible in this view, where the pink region marks one call to `m_vec_mul_add`, which consists of five limb operations, while the green region marks the sample length corresponding to one limb.

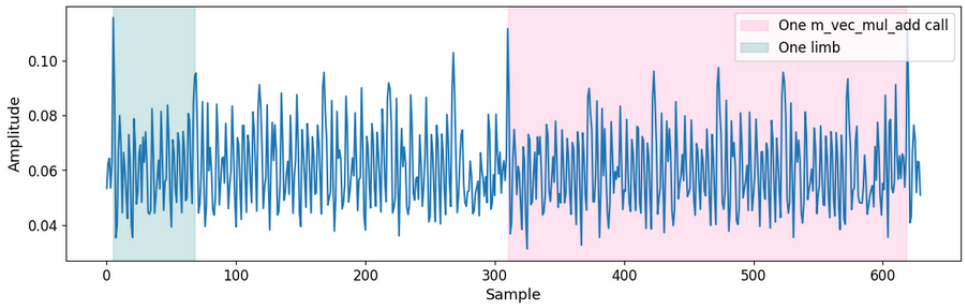



**Figure 4.4:** Two `m_vec_mul_add calls` from the 100-average trace.

### 4.1.2 Trace Processing

Figure 4.5 shows an overlay of all 77 segments corresponding to $\mathbf{O}[0, 0]$ extracted from the raw trace, where each segment has a length of 305 samples. As indicated by the differently colored spikes occurring at varying sample positions, the measurements contain a substantial amount of noise. Nevertheless, the segments align well with respect to the underlying trace pattern.

Figure 4.6 overlays the same 77 segments extracted from the trace averaged over 100 captures. Comparing the two figures illustrates how averaging preserves signal components that are consistent across captures, while reducing capture-specific noise.

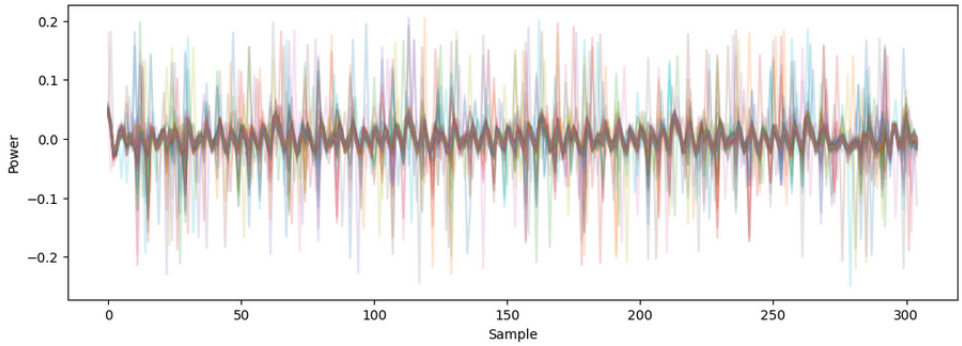



**Figure 4.5:** Raw trace overlay of 77 segments for one oil coefficient.

As a result, the averaged trace provides a clearer leakage profile, where the power pattern is more distinguishable as it is not affected by noise. However, averaging the 77 segments extracted from the raw trace reveals the same power pattern as each of the segments extracted from the averaged trace. This suggests that the noise present in the individual raw segments does not hinder the identification of PoIs, unless the segments are misaligned. Thus, as long as the peaks required for segmentation are available, the PoI identification is consistent between the raw and averaged traces. The advantage of the averaged trace is that the power values extracted at these points are less affected by noise, which provides cleaner input to the subsequent CPA.

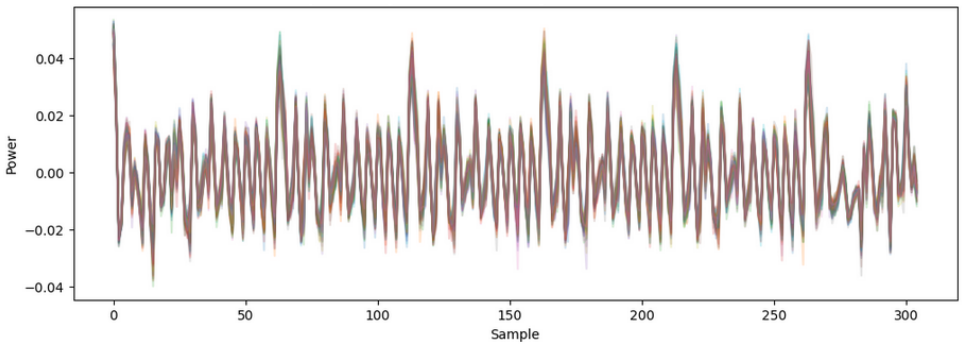



**Figure 4.6:** Average trace over 100 captures overlay of 77 segments for one oil coefficient.

*What is the minimum number of captures required to obtain an averaged trace that enables correct segmentation?* Recall from the previous section that the averaged trace was introduced to support segmentation of the full target-function trace into call-sized segments. This was done using a correlation-window scan, where correlation peaks were used to estimate the distance between repeated function calls. Once the peak positions had been identified, the same segmentation could be applied to both

the averaged trace and the raw trace. The initial segmentation was based on a trace averaged over 100 captures. However, to evaluate how much averaging is actually required, we investigated whether the number of captures could be reduced while still detecting all peaks necessary for correct segmentation. The minimum number of captures required to obtain an averaged trace that enables correct segmentation was found to be five. Averaging five captures does not reveal a repeating pattern as clearly as the trace averaged over 100 captures, but the underlying structure is still sufficiently visible to support segmentation. Compared with the 100-capture average, template selection is more difficult because the repeated pattern is less visually distinct. However, the experiments showed that an approximate template, obtained by dividing the total number of samples by the number of calls, was sufficient to identify all segments in the trace. This was the case even though the resulting segments did not all have identical lengths. This indicates that the correlation-scan technique does not require the exact call length to be known in advance. Instead, an approximate estimate is sufficient, which enables segmentation of traces where the repeated structure is less visually clear than initially assumed.

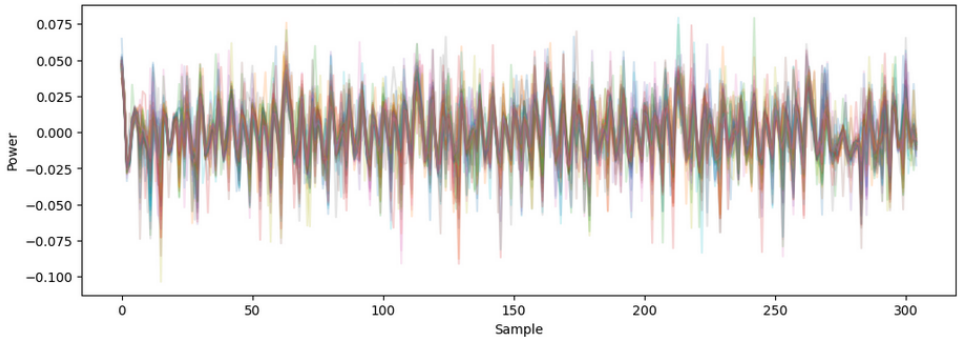



**Figure 4.7:** Average trace over 5 captures overlay of 77 segments for one oil coefficient.

An overlay of the 77 segments extracted from a trace averaged over five captures is shown in Figure 4.7. Compared with the raw trace overlay, a substantial amount of noise is reduced. However, the leakage profile is still noticeably less clear than in the trace averaged over 100 captures. The 5-capture average therefore represents an intermediate case between the raw trace and the 100-capture average. It was included as one of the three cases used in the remaining attack procedure to compare how different levels of averaging affect the attack results.

*Can the raw trace be segmented using an approximate call-length template extracted from the same raw trace?* Since the operation pattern is not visually distinguishable in the raw trace, an approximate call-length template must be used as the basis for the correlation scan. However, using such a template extracted from the raw

trace did not enable reliable segmentation of the same trace. The correlation output was not sufficiently clear to identify all correlation peaks. The results reported in `SCA-MQDSA` suggest that the correlation for the correct hypothesis can stabilize before all available leakage points for a secret value have been included [VGP+26]. This indicates that the attack does not necessarily rely on every available leakage point for each secret value. Nevertheless, preserving the overall segment structure remains necessary to correctly determine which secret value is updated at each point in the trace.

## 4.2 CPA Attack

The goal of the attack is for CPA to distinguish the correct key hypothesis from the other hypotheses by producing a higher correlation peak. We first present the CPA results for a single oil coefficient by displaying the resulting correlation for each hypothesis at the current position in the oil matrix. We then present the results for oil vector and oil matrix recovery.

### 4.2.1 CPA Results on a Single Oil Coefficient

The CPA results show how well the modeled intermediate values we have built the hypothesis from agree with the measured power consumption at the selected leakage points. The goal of the CPA on a single coefficient is for it to clearly, without ambiguity, point out the correct key hypothesis. As previously discussed, recovery of the next secret oil value in an oil vector requires the previous one to be distinguished correctly, since the accumulator must be updated by the true secret value between recoveries. Thus, a strong CPA result for a single coefficient is a prerequisite for the goal of recovering the entire oil matrix. The attack procedure, described in Section 3.4, was applied to three different power traces that capture the entire target function execution on STM32F4: the trace averaged over 100 captures, the raw trace, and the trace averaged over 5 captures. In each of the three cases, CPA is applied to the first oil position in an oil vector, which means that no previous updates have been made to the accumulator and it begins from the initial state in each recovery.

#### Average over 100 captures

The CPA was first applied to the trace that appeared to have the clearest leakage profile from the trace segmentation overlay for a single coefficient. The CPA results for the trace averaged over 100 captures, shown in Figure 4.8, show that the value 0 gives a correlation score greater than 0.8, while the other hypotheses have significantly lower correlations. From previous discussions we know that a Pearson correlation score close to zero indicates little or no correlation, while a value close to 1 in absolute value indicates strong correlation. In this case, 0 is identified as the correct key

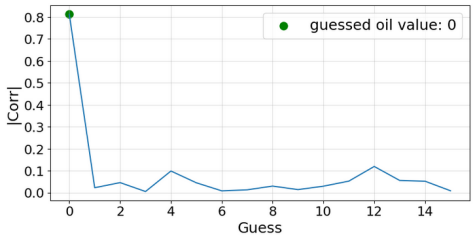



**Figure 4.8:** CPA results for 100-average trace, true oil value: 0.

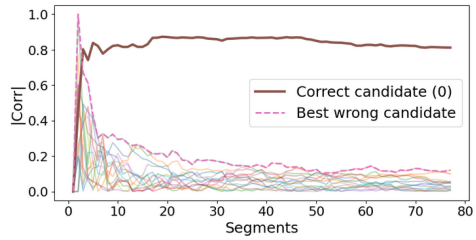



**Figure 4.9:** Correlation development over 77 segments for 100-average trace.

hypothesis because it produces the highest correlation score, and it is also the correct secret value for the oil matrix position under analysis.

Figure 4.9 shows the correlation development across all 77 segments, corresponding to 385 leakage points. The graph shows that the correlation starts to stabilize after approximately 10 segments. This means that only using 50 out of the 385 available leakage points is sufficient to obtain a reliable distinction between correct and incorrect key hypotheses under the current leakage profile.

However, this represents a best-case scenario, since the power trace under analysis contains little activity beyond the targeted cryptographic operations. For the STM32F4 target board, obtaining a trace this clear required averaging 100 signing procedures, which moves away from the single-trace scenario. At the same time, the noise level observed during trace capture is specific to the target board and measurement setup. The cleaner trace obtained from the SAM4S target board indicates that, under different conditions, a clear leakage profile may be achievable with fewer captures.

### Raw trace

Applying the CPA method to the raw trace gave the results shown in Figure 4.10. The correlation graph distinguishes the value 10 from the other hypotheses, with a correlation score just above 0.35. This is on the lower side of the Pearson correlation scale, but the comparison with the other hypotheses still shows a clear distinction. The guessed value 10 is the correct oil value for the position under analysis in the raw trace scenario, which means that CPA also recovers the correct value in this case. However, compared with the 100-capture average case, the raw trace produces a much lower correlation score and a smaller margin to the other hypotheses. This makes the result less robust than in the averaged case. This result is expected when considering the noisy segment overlay in Figure 4.5, together with the leakage model introduced in Equation (2.1). In this model, the noise term captures variation not explained by the modeled Hamming weight leakage. Such variation reduces the

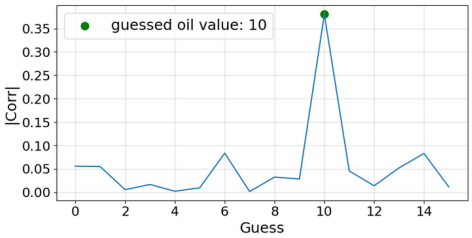



**Figure 4.10:** CPA results for raw trace, true oil value: 10.

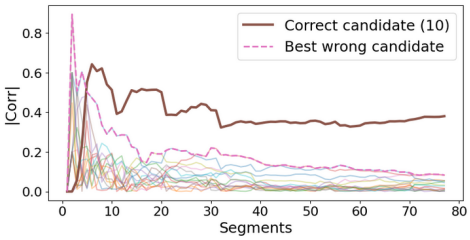



**Figure 4.11:** Correlation development over 77 segments for raw trace.

agreement between the measured and predicted leakage and can therefore explain the lower correlation observed for the raw trace.

Figure 4.11 shows how the correlation develops with the number of available segments for the raw trace. Although the correct key hypothesis reaches the highest correlation after approximately the same number of segments as in the 100-capture average case, the correlation does not stabilize until approximately 35 segments are used, corresponding to 175 leakage points. Thus, even though the correlation is weaker for the raw trace due to noise, the results show that CPA is still able to distinguish the correct hypothesis from the incorrect ones.

### Average over 5 captures

The third case applies the CPA to the trace averaged over five captures, which acts as an intermediate case between the two preceding cases since its leakage profile is clearer than that of the raw trace, but noisier than that of the trace averaged over 100 captures. The results in Figure 4.12 shows that the correct key hypothesis produces the highest correlation for this case as well. For the position under analysis, the correct secret value is 8, and the hypothesis for value 8 gives a correlation score just above 0.5, while the remaining hypotheses produce substantially lower correlations. As expected, this result lies between the two other cases. The correlation development in Figure 4.13 shows that the 5-capture average has a larger correlation margin to the other hypotheses than the raw trace. It also stabilizes around 10 segments, as in the 100-capture average case, but with a lower final correlation score.

Comparing the three cases shows that the correct key hypothesis is recoverable from all power traces, but with varying correlation strength. The results indicate that the amount of noise in the trace has a clear impact on the observed correlation score. As expected, the averaged traces produce stronger correlations than the raw trace, since averaging reduces capture-specific noise and results in a clearer leakage profile. An interesting observation is that, despite the higher noise level in the raw trace, the CPA is still able to distinguish the correct hypothesis from the incorrect ones. This

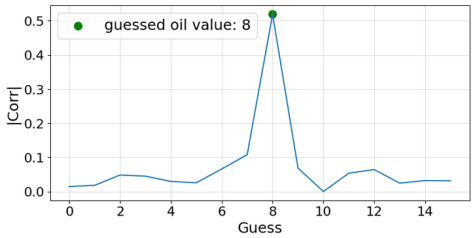



**Figure 4.12:** CPA results for 5-average trace, true oil value: 8.

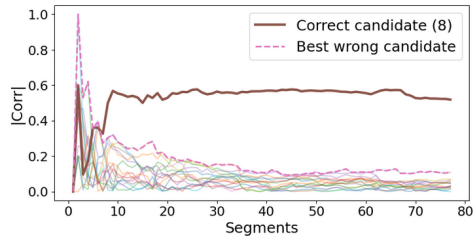



**Figure 4.13:** Correlation development over 77 segments for 5-average trace.

shows that the leakage remains exploitable even without averaging, although the distinction is weaker and less stable than in the averaged cases. Vishwaajith et al. report in `SCA-MQDSA` that the correct hypothesis reaches a correlation score between 0.5 and 0.6 in their single-trace experiments. This is comparable to the correlation observed in the 5-capture average case in our experiments. This suggests that the noise level and leakage profile of the 5-capture average may be closer to the raw single-trace measurements reported in `SCA-MQDSA` than to our raw STM32F4 trace.

### 4.2.2 CPA on Full Oil Vector

For recovery of a full oil vector, the CPA method is applied for one oil vector row at a time. After each coefficient is recovered, the accumulator is updated with the guessed value before CPA is applied to the next coefficient. The lookahead mechanism used in oil-vector recovery is controlled by two parameters, namely the `ambiguity_margin`, which determines how large the gap between the highest and second-highest correlation scores must be for a result to be accepted directly, and the `lookahead_top_k`, which determines how many of the top-ranked candidates are considered during the lookahead step. In the following experiments, `lookahead_top_k` is set to 3, while different ambiguity margins are tested to evaluate how much the recovery of a full oil vector depends on the lookahead mechanism in each of the three trace cases. The ambiguity margin is defined as a threshold on the difference between the best and second-best candidate, rather than as an absolute minimum correlation score. This allows the best candidate to be accepted even when the overall correlation is reduced by noise, as long as it remains sufficiently separated from the competing hypotheses.

Table 4.1 shows the number of times the lookahead mechanism was used during recovery of a single oil vector for different ambiguity margins. The table also shows whether the full oil vector was recovered correctly for each of the three trace cases. For the 100-capture average, the oil vector was recovered correctly for all tested ambiguity margins without using the lookahead mechanism. This indicates that the leakage profile is sufficiently clear for the correct hypothesis to be accepted directly

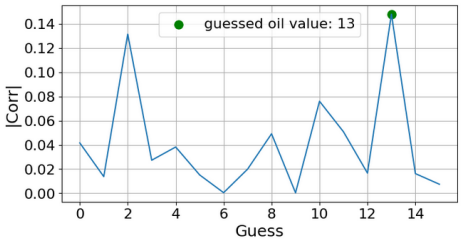
**Table 4.1:** Lookahead (LA) count and recovery outcome for different ambiguity margins during recovery of a single oil vector for the three different traces.

| | Raw trace | | 5-average | | 100-average | |
|---|---|---|---|---|---|---|
| Margin | LA count | Recovered? | LA count | Recovered? | LA count | Recovered? |
| 0.0 | 0 | × | 0 | ✓ | 0 | ✓ |
| 0.15 | 35 | ✓ | 0 | ✓ | 0 | ✓ |
| 0.25 | 59 | ✓ | 2 | ✓ | 0 | ✓ |
| 0.30 | 75 | ✓ | 10 | ✓ | 0 | ✓ |
| 0.35 | 76 | ✓ | 37 | ✓ | 0 | ✓ |

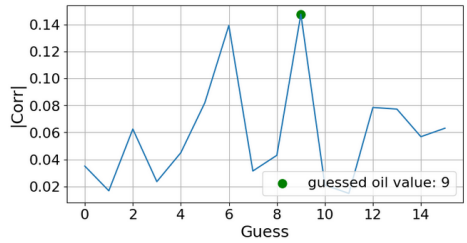
throughout the recovery. In contrast, the raw trace fails when the ambiguity margin is set to 0.0, meaning that always selecting the highest-correlation hypothesis is not sufficient in this case. When the ambiguity margin is increased to 0.15, the vector is recovered correctly, but the lookahead mechanism is used 35 times. This shows that several coefficient recoveries in the raw trace are ambiguous and require additional verification. The 5-capture average lies between the two other cases. It is recovered correctly for all tested margins, and no lookahead is required for margins up to 0.15. However, as the margin is increased, the number of lookahead uses also increases. This reflects the intermediate leakage profile observed earlier, where averaging over five captures reduces noise compared with the raw trace, but does not provide a leakage profile as clear as the 100-capture average.

As previously discussed, the 100-average capture was included as an initial case to enable trace segmenting. However, it is useful to determine the minimum number of averaged traces required for the lookahead count to reach zero for all ambiguity margins listed in Table 4.1. By repeating the attack with different numbers of averaged traces, we found that an average of 10 traces was sufficient to reduce the lookahead count to zero for all ambiguity margins up to 0.35.

The lookahead mechanism acts as a security net in cases where the CPA result is not sufficiently clear. Figures 4.14a and 4.14b show examples of low-margin recoveries from the raw trace where the correct hypothesis still produces the highest correlation. However, the separation from the next-best hypothesis is very small, with margins of 0.017 and 0.002, respectively. Figures 4.15a and 4.15b show examples where the direct CPA decision selects the wrong hypothesis during oil vector recovery. In these cases, the highest correlation peak does not correspond to the correct oil value. This demonstrates why accepting the hypothesis with the highest correlation without an ambiguity criterion can lead to incorrect recovery.

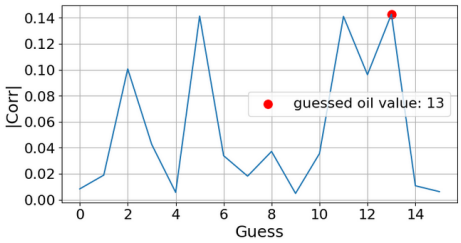



**(a)** Correct hypothesis selected, correlation margin to next hypothesis: 0.017.

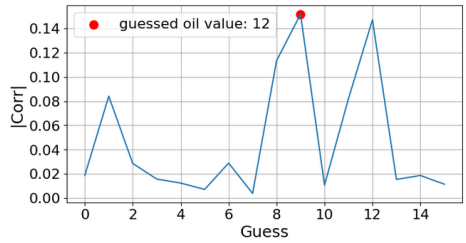



**(b)** Correct hypothesis selected, correlation margin to next hypothesis: 0.002.

**Figure 4.14:** CPA examples from the raw trace with small margins between the best and second-best hypotheses.




**(a)** Incorrect hypothesis selected, correct value: 11.




**(b)** Incorrect hypothesis selected, correct value: 12.

**Figure 4.15:** CPA examples from the raw trace where the highest correlation hypothesis does not correspond to the correct oil value.

Overall, the results show that the ambiguity margin controls how conservative the recovery procedure is. A higher margin increases the number of cases that are treated as ambiguous and passed to the lookahead mechanism. This improves robustness against uncertain CPA decisions, but also increases the amount of additional verification required. The raw trace is most affected by this trade-off, since noise reduces the separation between the best and second-best hypotheses. Generally, the higher the ambiguity margin, the more certain we can be in the correctness of the results.

### 4.2.3 CPA on Full Oil Matrix

Applying the oil vector recovery procedure to each of the eight vectors in the oil matrix **O** allowed the entire matrix to be recovered correctly for all three trace cases considered. The method was also evaluated on three different datasets, where all 624 secret values of the oil matrix were recovered correctly in each case.

## 4.3 Countermeasure Evaluation

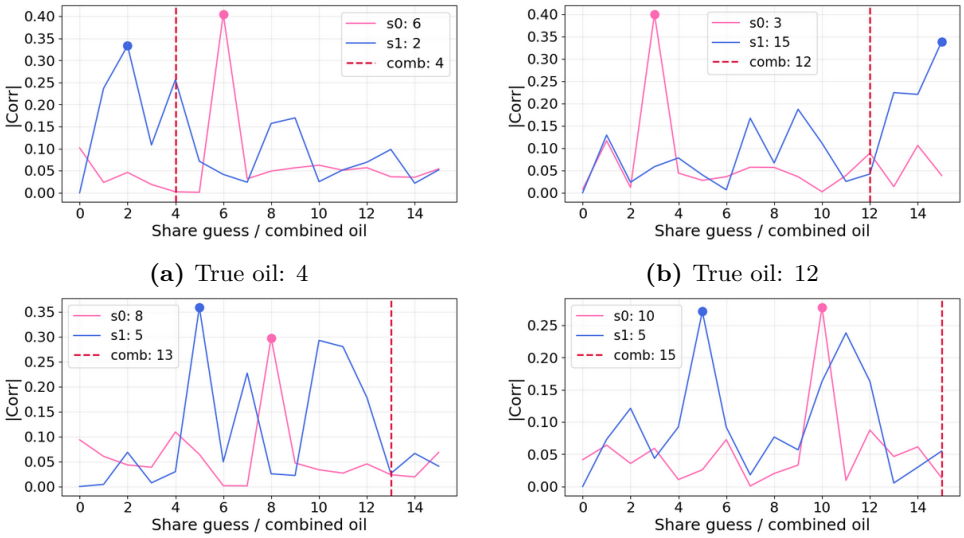
In this section, we present the results and analysis of applying the CPA attack to the masked implementation of MAYO using two and three shares. For the two-share case the attack is applied to the raw trace, while for the three-share case we use the 5-capture average trace for clearer distinguishing in the resulting graphs.

### 4.3.1 Two Shares

Applying the CPA attack to the masked implementation of MAYO with two shares gives the results shown in Figure 4.16. The figure shows the per-share CPA results for four coefficients in the first row of the oil matrix. For each of these, the two recovered share values combine to the correct secret oil value, marked in dotted pink in each sub-figure. The results show that we can recover the masked representation by attacking the two shares separately. However, the distinguishing is not equally strong for the two shares. In several cases, the second share, share 1, gives more ambiguous correlation peaks than share 0. In the implementation used here, the shares are randomly selected for each iteration, so the attack must be performed on the raw trace, which remains noisy also for the two-share setting. However, as shown in the previous attack, this type of ambiguity can be handled using the lookahead mechanism.

Applying the oil vector recovery procedure and the lookahead mechanism separately to both shares results in two recovered share vectors. The bitwise XOR of these vectors gives the corresponding secret oil vector. Extending this procedure to the full oil matrix allowed all 624 secret values to be recovered, even in the presence of two shares. The full matrix recovery was performed using an ambiguity margin of 0.2 and `lookahead_top_k` equal to 3. Other parameter choices also resulted in full recovery, indicating that result is not dependent on a single narrow parameter setting.

*Why does share 1 generally give weaker distinguishing than share 0?* Although the same type of operation is performed on both shares, the observed correlation is generally more ambiguous for share 1. One possible explanation is the difference in accumulator initialization. The accumulator for share 0 is initialized from **P2**, while the accumulator for share 1 starts from zero. This changes the intermediate accumulator values and may therefore affect the leakage profile. This factor could explain why share 1 is harder to distinguish in several cases, although further experiments would be required to confirm this explanation.




**(a)** True oil: 4

**(b)** True oil: 12

**(c)** True oil: 13

**(d)** True oil: 15

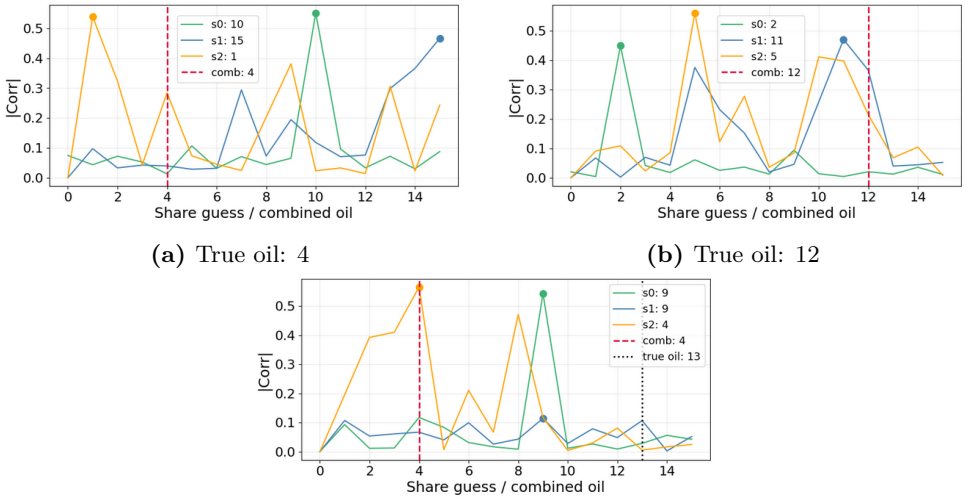
**Figure 4.16:** CPA results for the last four coefficients in the first row of the oil matrix for the masked MAYO implementation with two shares.

### 4.3.2 Three Shares

For the three-share implementation, the attack must be applied independently to all three shares before the recovered values can be recombined. To reduce the effect of noise and make the leakage easier to evaluate, the three-share experiment was performed using fixed shares and an average of five traces. Applying the same share-wise CPA method to the average three-share trace showed that the leakage is still present. Each share could be attacked separately, and the recovered share values could then be XORed to reconstruct the oil coefficients. Figure 4.17 shows the share-wise CPA results for three shares and their combined oil value, which corresponds to the true secret oil in Figures 4.17a and 4.17b, while Figure 4.17c gives the wrong combined result. By incorporating some modifications based on observations made during the recovery process, the application of the oil vector recovery procedure with the lookahead mechanism made it possible to recover the secret values of the entire oil matrix.

#### Observations during recovery

During the recovery, a special case was observed when both an accumulator share and the corresponding oil share were zero. In the implementation, the first accumulator share is initialized with **P2**, while the remaining accumulator shares are initialized to zero, such that $\text{acc}_0 = \mathbf{P2}$, $\text{acc}_1 = 0$, and $\text{acc}_2 = 0$.




**(a)** True oil: 4

**(b)** True oil: 12

**(c)** Incorrect combination, True oil: 13

**Figure 4.17:** CPA results for three shares on three oil matrix positions in row 0.

In Figure 4.17c, the blue curve, corresponding to share 1, shows weak and poorly separated correlation peaks. As a result, the share value is recovered incorrectly, which in turn leads to an incorrect recombined oil value. The true value for share 1 in this case is zero. When either share 1 or share 2 has value zero in the first row, the corresponding hypothesis may appear almost constant in the leakage model and therefore provide little variation for the CPA distinguisher to exploit. This may explain why the distinguisher fails to produce a clear peak and why neither direct CPA nor the lookahead mechanism is sufficient in this case. This effect was observed for share 1 and share 2, but not in the same way for share 0. A likely explanation is that share 0 is initialized with **P2**, which still provides variation in the leakage model even when the share value is zero.

This behavior was first observed during recovery of the full oil matrix, where an entire oil vector was not recovered correctly. Further investigation showed that recovery failed in the first row of that vector. Since the accumulator state is updated row by row, this incorrect decision at the first row, caused the remaining entries in the vector to be recovered incorrectly as well.

To verify whether the 0 value could explain the behavior, the fixed shares were regenerated so that a zero value occurred in a different first-row position for share 1 or share 2. After rerunning the attack, the recovery failure moved to the new position where the same condition was present. Conversely, the vector that previously failed was recovered correctly once its first-row non-$P2$-initialized shares were no longer

zero. This supports the hypothesis that the weak correlation is caused by a weak leakage model in the specific case where $z = 0,\ \text{share} \in \{1, 2\},\ \mathbf{O}_{\text{share}}[0, k] = 0$. Since this effect appears to occur before any non-zero contribution has been accumulated into the corresponding accumulator share, it is mainly relevant at the beginning of each oil-vector recovery. In our experiments, it was observed in the first row of an oil vector. However, the same issue could in principle persist for later rows if the same share is zero for the first row and for one or more consecutive rows afterwards. In that case, the accumulator share may remain close to its initial zero state, and the leakage model may still provide too little variation for the CPA distinguisher. Once a non-zero contribution has been accumulated into the share state, this effect is expected to disappear.

To overcome this edge case, we introduced a simple threshold rule. If the recovery is performed on share 1 or share 2, the row index is zero, and the maximum absolute correlation is below a fixed threshold, the share value is assumed to be zero. In the experiments presented here, it was sufficient to handle only the first-row case. In the experiments, we used a threshold of 0.12, which allowed the recovery to proceed correctly through the first row. With this modification, the complete oil matrix was recovered for the three-share implementation.

In our implementation, **P2** is placed in the first accumulator share before the multiplication is performed. This was chosen because it is closest to the structure of the original unmasked implementation, where the accumulator is initialized with **P2**. However, in the masked setting this introduces an asymmetry between the shares, which may be undesirable from a side-channel perspective. A more symmetric implementation could initialize all accumulator shares to zero during the multiplication and add **P2** to one share only after the multiplication has completed, as in [KNK+25, p. 9]. In that case, the first-row zero-share observation discussed above would likely also apply to share 0.

## 4.4 Summary of Results

The results presented in this chapter show that the target function of the MAYO implementation leaks sufficient information to recover the secret oil matrix. The initial power measurements demonstrate that the repeated structure of `m_vec_mul_add` calls is clearly visible in the raw SAM4S trace, and becomes visible in the full STM32F4 trace after averaging. This structure made it possible to segment the trace into calls, group them by coefficients, and identify leakage points for CPA.

The CPA results for a single coefficient show that the correct oil coefficient can be distinguished from the other hypotheses in all three trace settings considered, namely the 100-capture average, the 5-capture average (which provides the minimum

average captures needed for correct segmentation), and the raw trace. As expected, averaging improves the correlation strength and makes the leakage profile clearer. One important finding is that the raw trace contains exploitable leakage despite the high level of noise. This indicates that the main limitation is not the absence of leakage, but the difficulty of obtaining accurate segmentation under noisy measurement conditions. The oil vector and full matrix recovery experiments further show that the attack scales from an individual coefficient to the complete oil matrix, where the lookahead mechanism improves robustness when the CPA result is ambiguous, especially for the raw trace where the separation between the best and second-best hypothesis is often small. With this mechanism, the full oil matrix was recovered correctly for all three trace cases and across multiple datasets.

The countermeasure evaluation shows that masking increases the complexity of the attack, but does not prevent recovery of the oil matrix in the tested implementation. For both two and three shares, the shares could be attacked separately and recombined to recover secret oil values. This shows that masking alone is not sufficient to protect from the attack if the adversary can correctly segment the trace and group leakage by share and coefficient. The three-share case also revealed an implementation-specific edge case, where a zero share in the first row produces very weak correlations for accumulator shares initialized with zero. After accounting for this case, the full oil matrix could also be recovered for the three-share implementation.

# Chapter 5
# Discussion

In this chapter, we discuss the implications and limitations of the experimental results presented in Chapter 4. We consider the practical limitations encountered during reproduction of the attack, with particular focus on the role of trace segmentation. We further discuss the observations made during the masking evaluation, before briefly considering transferability to other multivariate schemes. Finally, we discuss the implications of the results for side-channel protection, including how the attack may be complicated for an adversary and directions for future research.

## 5.1 Practical Limitations and Trace Segmentation

A central consideration throughout this work has been how to evaluate the attack under realistic conditions. The initial expectation was that a single trace would be sufficient for a successful attack, including trace segmentation and correlation analysis. In practice, several challenges made this ideal scenario difficult to achieve. One target board introduced memory constraints due to the large parameter sizes of MAYO, while the board able to execute the full target function produced noisy traces that did not allow reliable segmentation directly. We thus encountered two suboptimal scenarios, where simplifying the function to work around memory limitations moves the experiment away from a complete real-world execution, while averaging traces to obtain a cleaner signal moves it away from a strict single-trace setting. Both approaches were therefore used as tools to understand different aspects of the leakage.

The noisy trace should nevertheless not be dismissed as unrealistic. In a real implementation, power measurements may also be affected by surrounding operations, which can disturb the visual structure of the target operation. In that sense, the noisy traces may provide useful insight into a realistic measurement scenario. The results show that the main obstacle in this setting is not necessarily the absence of leakage, but the difficulty of correctly segmenting the trace. In the experiments, this was addressed by averaging several operations in order to obtain a sufficiently clear reference trace for segmentation.

For the unmasked implementation, collecting multiple observations may also be a realistic assumption as the secret key expansion is deterministic for a fixed key. This means that the same secret-dependent computation is repeated whenever the expanded key is generated for the same key holder, which allows for averaging. The experiments show that only a small number of observations is needed to obtain a leakage profile clear enough for segmentation and full recovery. The small number of required observations is what makes the attack strong, where under cleaner measurement conditions, a single trace is sufficient, while in noisier settings, only a few repeated traces are needed for our methodology.

The results therefore suggest that noise mainly affects the attack by making trace segmentation harder, rather than by removing the leakage itself. This is supported by the fact that the entire oil matrix can be recovered from a raw trace once accurate segments have been obtained from an average trace. The attack is therefore highly dependent on precise segmentation, and the segmentation procedure we used was not sufficient to split the raw trace directly. However, other segmentation techniques may be able to perform this step even in noisier traces. The `SCA-MQDSA` article provides limited detail on this part of the process, possibly because the traces shown there were cleaner and therefore did not counter the same problems on this part.

In the masked implementation, the role of averaging is different. If fresh randomness is used to generate new shares for each execution, averaging traces no longer preserves the same intermediate share values. In that case, averaging cannot be used in the same way to average out noise, but can still be useful for identifying the general operation structure to enable trace segmentation.

## 5.2 Zero-Value Shares as Leakage Indicators

The zero-share edge case observed during masking evaluation illustrates that weak CPA separation does not necessarily mean that no information is leaked. In the three-share experiment, recovery failed when both the initial accumulator share and the contribution from the corresponding first-row oil share were zero, causing the modeled accumulator update to provide little variation for the CPA distinguisher. If this situation is not detected, an incorrect decision in the first row of an oil vector may corrupt the accumulator state and cause the remaining coefficients in the vector to be recovered incorrectly. From the adversary's perspective, however, this behavior can also be informative. Once the edge case is understood, a weak or nearly flat correlation result may itself indicate that the corresponding share value is zero. Conversely, if a clear correlation peak is observed for the same accumulator-share setting, the adversary may exclude zero as a likely value and restrict the candidate space to the remaining non-zero field elements. If the masked oil coefficients for the same share index are zero in several consecutive rows, the same behavior may persist

until a non-zero contribution is accumulated into the corresponding accumulator share. Thus, the special case may leak information even though the CPA distinguisher does not rank the correct hypothesis highest in the ordinary sense.

Although in a different cryptographic setting, prior side-channel work has shown that zero values in intermediate computations can be useful from an attacker's point of view. For instance, Goubin introduced a power-analysis attack that exploits special zero-value points in elliptic curve implementations [Gou02], and Akishita and Takagi later extended this idea by considering zero-value registers that may arise during elliptic curve computations [AT03]. The setting considered here is different, but the general lesson is relevant. Exceptional intermediate values can create recognizable leakage behavior that should be considered during side-channel evaluation.

## 5.3 Transferability to Other DSAs

In the pre-project, we motivated the choice of the multivariate family by aiming to investigate a target that could provide insight beyond a single isolated algorithm. At the time, several candidates were based on the multivariate UOV construction which made this family an interesting choice for studying whether the findings could have broader relevance. Although the transferability of a specific attack is affected by factors such as structural design, underlying arithmetic fields, and implementation choices, Vishwaajith et al. argue in `SCA-MQDSA` that their approach is not specific to MAYO [VGP+26]. They extend their analysis to UOV and indicate that the attack framework is also applicable to related schemes such as QR-UOV and SNOVA. Together with MAYO, these schemes are among the multivariate candidates that advanced to the third round of the standardization process [Nat26]. This further motivates the need for side-channel protection against the type of attack studied in this thesis.

## 5.4 Side-Channel Protection and Future Research Directions

The countermeasure evaluation shows that introducing masking by splitting the secret values into shares does not by itself prevent recovery of the full oil matrix. This was observed for both two and three shares. If the attacker can correctly group the leakage points by share and coefficient, the masked attack can be reduced to several instances of the original recovery problem, followed by recombination of the recovered shares. In this sense, the statistical problem remains similar, but the attacker must solve it separately. Masking therefore increases the complexity of the attack, but it is not sufficient as a standalone countermeasure against the attack strategy considered in this thesis.

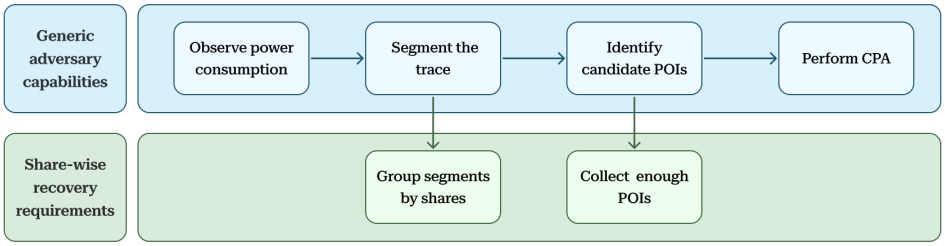



**Figure 5.1:** Adversary capabilities and requirements for successful CPA on masked MAYO.

The question now is how we can make the attack as difficult as possible under the current implementation. Figure 5.1 illustrates the assumptions required for the share-wise attack on the masked implementation to succeed. The upper part of the figure represents general adversary capabilities, while the lower part highlights the requirements for a successful attack on the masked implementation. In particular, the adversary must be able to group trace segments by share and coefficient, and must collect enough leakage points per share to obtain reliable correlation results. These two requirements make natural targets for countermeasures as we can either make it harder to collect enough useful leakage for each share, or make it harder to determine which leakage belongs to which share and coefficient.

Assuming correct trace segmentation, hindering the collection of enough leakage points is difficult in the single-trace setting. By splitting the secret values into shares, we add randomness to the procedure, which keeps the adversary from collecting leakage points from multiple signing executions. However, the masking randomness is refreshed only between executions. It therefore does not limit the number of leakage points available within a single trace and since a single trace already contains enough leakage for a successful attack, preventing this would require refreshing the shares during the computation. From the experiments, the correlation often stabilizes after roughly 50 leakage points. Since each oil coefficient is involved in 385 leakage points, this suggests that re-masking would need to be applied multiple times during the computation to substantially reduce the number of useful observations per share. For example, reducing the number of observations below this empirical threshold would require approximately eight refreshes per coefficient. Such an approach would increase both implementation complexity and execution time. It is therefore not necessarily a practical countermeasure for the current implementation, but it may point toward considerations that could be useful in designing more side-channel-aware matrix multiplication routines.

The other main requirement is the ability to group segments by share and coefficient. As pointed out in earlier discussions, the attack is fully dependent on knowing which

parts of the trace correspond to which share, and which of those operations involve a specific oil coefficient. In our implementation, all operations for one share are performed before moving to the next share. This deterministic ordering makes grouping by share straightforward and therefore leaves exploitable structure in the trace. Hiding is a countermeasure that targets exactly this. The purpose of hiding is to obscure the relationship between trace segments and sensitive operations by randomizing the physical output of the device and thereby making it harder to exploit the leakage present.

Segmentation is feasible due to the recurring call pattern of the target operation. However, this same pattern may also be exploited by the protector. Since the patterns across individual calls are not directly distinguishable in the power trace, randomly shuffling the order of computations may be an effective countermeasure against the grouping procedure, as the execution order can no longer be inferred directly from the code. Thus, after randomly shuffling, the attacker can no longer rely on the code structure to determine which observed segment corresponds to which share or coefficient. Since the computation is linear and each share contributes independently to the final recombined value, the order in which the share computations are performed should not affect the correctness. The only requirements are that each contribution is accumulated into the correct accumulator share and that **P2** is added to exactly one share. Also, the effect of shuffling is expected to increase with the number of shares, since the number of possible share orderings grows. Thus, with more shares, the attacker must solve a larger grouping problem before the share-wise CPA can be applied. Though this may be an effective countermeasure against the attack methodology used in this thesis, a more powerful adversary may be able to perform correct grouping. Nevertheless, it targets one of the main assumptions used in the attack and could therefore be a useful complement to masking.

Other hiding techniques include dummy operations and jitter. Jitter introduces random delays during execution, which mainly makes it more difficult for the attacker to align traces accurately. If the jitter is only placed between calls, the repeated call structure may still be identifiable through a correlation-based segmentation procedure. If jitter is inserted within calls, the internal call pattern may vary, making it harder to align segments and identify common leakage points across segments, which is also a crucial part of the attack. Dummy operations could also make grouping more difficult, especially if they are inserted unpredictably and have a trace structure similar to the actual `m_vec_mul_add` operation. This could largely disrupt the grouping by coefficient, since uncertainty about the placement of even a single dummy operation could desynchronize the process of associating segments with coefficients. These discussions suggest that applying protective countermeasures to the earlier stages of the attack may hinder the attacker from collecting a sufficient amount of leakage points, even though they are present.

A general challenge with all of these countermeasures is the trade-off between protection and implementation cost. Increasing the number of shares increases memory usage, while shuffling requires scheduling logic and random choices during execution. Similarly, dummy operations and jitter increase execution time. This trade-off is worth considering in the context of MAYO, since one of its motivations is to reduce the main practical drawback of traditional UOV-based schemes, the large public key size. This improvement makes MAYO more attractive for practical deployment, including on resource constrained devices, but countermeasures such as masking and hiding techniques can introduce overhead that may reduce some of these practical advantages. More experiments would be needed to quantify the actual implementation cost of the different countermeasures. However, the targeted matrix multiplication may be side-channel protection-friendly since it mainly consists of linear operations.

Overall, the results suggest that masking alone is not sufficient to protect against the single-trace attack considered in this thesis. However, masking can still serve as a useful foundation when combined with hiding countermeasures that make share-wise grouping more difficult, which in turn hinders the collection of leakage points. This would not eliminate the existing leakage, but it makes it harder for the attacker to exploit the existing leakage. These observations suggest that countermeasures that disturb the early trace-processing assumptions of the attack may be particularly promising for this single-trace attack. Therefore, experiments on the effect of combinations of hiding on top of masking would be interesting future research directions. Another interesting direction is to investigate alternative implementations of the matrix multiplication with more side-channel awareness, which could help determine whether the leakage could be reduced at the implementation level.

# Chapter 6
# Conclusion

In this thesis, we have investigated the side-channel security of multivariate post-quantum digital signatures, using nibble-sliced MAYO as the concrete target for experimental evaluation. We focused on power-based leakage from a secret-dependent operation in the MAYO secret key expansion and studied whether this leakage could be exploited using non-profiled CPA, following the attack strategy introduced by Vishwaajith et al. [VGP+26]. In addition, masking was evaluated as an implementation-level countermeasure by attacking masked versions of the targeted operation with two and three shares.

> **RQ1** *To what extent are multivariate post-quantum digital signature algorithms from the NIST second round susceptible to side-channel leakage?*

We have not evaluated all multivariate candidates, but have studied the question through MAYO. Within this scope, the results show that the targeted MAYO implementation leaks sufficient information through power consumption to recover the secret oil matrix. The attack succeeds, and the full oil matrix was recovered for all three trace settings considered: the 100-capture average, the 5-capture average, and the raw trace. This demonstrates that, for the evaluated implementation and attack model, the leakage from the targeted operation is strong enough to compromise the signature scheme. Since the targeted operation is related to arithmetic structures that also appear in other multivariate schemes, the findings may also provide insight into side-channel concerns for other candidates in the NIST additional signatures process. However, the exact transferability depends on the structure and implementation of each scheme.

**RQ2** *Which specific suboperations within multivariate-based digital signature algorithms exhibit the highest susceptibility to side-channel leakage under practical attack scenarios?*

The analysis of **RQ2** centers on the key-expansion operation in MAYO, where public matrix data are multiplied with the secret oil matrix. The results show that this operation is highly susceptible to side-channel leakage in the evaluated implementation. Since the same multiplication routine is repeated many times for each secret oil coefficient, the attacker obtains enough leakage points within one execution to apply CPA. Although the thesis does not establish that this is the most vulnerable operation among all possible candidates, the experiments show that it exhibits a high degree of susceptibility under the practical attack scenario considered. As for **RQ1**, these findings may provide insight into side-channel vulnerabilities in other multivariate-based digital signature algorithms as well.

**RQ3** *How effective is a ChipWhisperer-based setup in capturing side-channel leakage from multivariate digital signature implementations?*

ChipWhisperer provided a useful and effective platform for the experimental analysis in this thesis. The open-source toolchain and documentation included clear guidance on firmware modification, custom firmware experiments, and troubleshooting. At the same time, the experiments show that the effectiveness of such a setup depends strongly on the target board and the practical constraints of the implementation. The SAM4S setup provided clear traces for isolated parts of the attack, but did not have sufficient memory to execute the full target function. The STM32F4 setup could execute the full target operation, but the measured traces were significantly noisier. Averaging made the repeated operation structure visible and enabled segmentation, while the raw trace still contained exploitable leakage once accurate segmentation information was available. This suggests that the ChipWhisperer-based setup is effective for capturing leakage from MAYO, but that memory limitations are a practical challenge when working with PQC implementations. It is possible that improved measurement settings, board configuration, or alternative segmentation techniques could reduce the noise challenge observed on the STM32F4, but this was not fully resolved within the scope of this thesis.

**RQ4** *What implementation-level countermeasures appear most effective in mitigating side-channel leakage in multivariate digital signature schemes, and how can their efficacy be evaluated through experimental analysis?*

For countermeasure evaluation we focused on Boolean masking with two and three shares. The results show that masking increases the complexity of the attack, but does not prevent recovery in the tested implementation. For both two and three shares, the

shares could be attacked separately and recombined to recover the secret oil values. This indicates that share-wise masking alone is not sufficient under the considered attack model when the adversary can segment the trace, group leakage by share and coefficient, and collect enough leakage points per share. The broader question of which countermeasures are most effective was not fully answered experimentally, since hiding techniques such as shuffling, dummy operations, and jitter were not implemented and tested. However, the results suggest that countermeasures that make trace segmentation and share grouping more difficult may be especially relevant for this attack strategy. More generally, the effectiveness of such countermeasures can be evaluated by measuring whether they reduce exploitable correlation, prevent reliable trace segmentation, or prevent full oil-matrix recovery.

In conclusion, this thesis shows that implementation security is a critical part of evaluating post-quantum digital signatures. A scheme may be designed to resist classical and quantum cryptanalysis, but still be vulnerable when implemented on physical hardware. The evaluated MAYO implementation demonstrates this, as secret-dependent matrix operations leaked enough information through power consumption to recover the secret oil matrix. For multivariate schemes such as MAYO, protecting these operations against side-channel attacks is therefore important to build confidence in practical deployment. The results further suggest that masking may need to be considered together with other countermeasures, such as hiding, or adapted implementation strategies when protecting multivariate matrix multiplication against single-trace side-channel attacks.

# References


[ABC+24] G. Alagic, M. Bros, et al., "Status Report on the First Round of the Additional Digital Signature Schemes for the NIST Post-Quantum Cryptography Standardization Process", National Institute of Standards and Technology, NIST Interagency/Internal Report NIST IR 8528, Oct. 2024. DOI: 10.6028/NIST.IR.8528.

[ADKF70] V. L. Arlazarov, Y. A. Dinitz, et al., "On Economical Construction of the Transitive Closure of an Oriented Graph", *Doklady Akademii Nauk SSSR*, vol. 194, pp. 487–488, 1970. [Online]. Available: http://mi.mathnet.ru/dan35675.

[AT03] T. Akishita and T. Takagi, "Zero-value point attacks on elliptic curve cryptosystem", in *Information Security*, C. Boyd and W. Mao, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 218–233, ISBN: 978-3-540-39981-0.

[Bar09] G. V. Bard, "The method of four russians", in *Algebraic Cryptanalysis*. Boston, MA: Springer US, 2009, pp. 133–158, ISBN: 978-0-387-88757-9. DOI: 10.1007/978-0-387-88757-9_9.

[BBD09] D. J. Bernstein, J. Buchmann, and E. Dahmen, Eds., *Post-Quantum Cryptography*, 1st ed. Berlin, Heidelberg: Springer, 2009, ISBN: 978-3-540-88702-7. DOI: 10.1007/978-3-540-88702-7.

[BCC+24] W. Beullens, F. Campos, et al., "Nibbling MAYO: Optimized Implementations for AVX2 and Cortex-M4", *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2024, no. 2, pp. 252–275, 2024, Artifact available at https://artifacts.iacr.org/tches/2024/a14. DOI: 10.46586/tches.v2024.i2.252-275.

[BCC+25] W. Beullens, F. Campos, et al. "MAYO Specification Document - Round 2", Accessed: Apr. 13, 2026. [Online]. Available: https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-2/spec-files/mayo-spec-round2-web.pdf.

[BCH+23] W. Beullens, M.-S. Chen, et al., "Oil and Vinegar: Modern Parameters and Implementations", *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2023, no. 3, pp. 321–365, 2023. DOI: 10.46586/tches.v2023.i3.321-365.

[BCO04] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model", in *Cryptographic Hardware and Embedded Systems - CHES 2004*, M. Joye and J.-J. Quisquater, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 16–29, ISBN: 978-3-540-28632-5.

[Beu22] W. Beullens, "MAYO: Practical post-quantum signatures from oil-and-vinegar maps", in *SAC 2021: 28th Annual International Workshop on Selected Areas in Cryptography*, R. AlTawy and A. Hülsing, Eds., ser. Lecture Notes in Computer Science, vol. 13203, Virtual Event: Springer, Cham, Switzerland, Sep. 2022, pp. 355–376. DOI: 10.1007/978-3-030-99277-4_17.

[DPS20] J. Ding, A. Petzoldt, and D. S. Schmidt, *Multivariate Public Key Cryptosystems* (Advances in Information Security), 2nd ed. New York, NY: Springer, 2020, ISBN: 978-1-0716-0987-3. DOI: 10.1007/978-1-0716-0987-3.

[DRC+25] P. Dobias, A. Rezaeezade, et al., *SoK: Reassessing side-channel vulnerabilities and countermeasures in PQC implementations*, Cryptology ePrint Archive, Paper 2025/1222, version: 20250630:124712, 2025. [Online]. Available: https://eprint.iacr.org/2025/1222.

[Eur23] European Union Agency for Cybersecurity (ENISA). "Post-quantum cryptography: Current state and quantum mitigation (v2)", Accessed: Oct. 1, 2025. [Online]. Available: https://www.enisa.europa.eu/sites/default/files/publications/ENISA%20Report%20-%20Post-Quantum%20Cryptography%20Current%20state%20and%20quantum%20mitigation-V2.pdf.

[Eur25] European Commission. "EU reinforces its cybersecurity with post-quantum cryptography", Accessed: May 28, 2026. [Online]. Available: https://digital-strategy.ec.europa.eu/en/news/eu-reinforces-its-cybersecurity-post-quantum-cryptography.

[FS] M. Flinders and I. Smalley. "Firmware vs. software: What's the difference and why does it matter?", IBM, Accessed: Apr. 11, 2026. [Online]. Available: https://www.ibm.com/think/insights/firmware-vs-software.

[GNU] GNU Project. "Flags", Accessed: Apr. 20, 2026. [Online]. Available: https://gcc.gnu.org/onlinedocs/gccint/Flags.html.

[Gou02] L. Goubin, "A refined power-analysis attack on elliptic curve cryptosystems", in *Public Key Cryptography — PKC 2003*, Y. G. Desmedt, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 199–211, ISBN: 978-3-540-36288-3.

[Ive25] M. Ivezic. "Q-Day Revisited – RSA-2048 Broken by 2030: Detailed Analysis", Accessed: May 28, 2026. [Online]. Available: https://postquantum.com/q-day/q-day-y2q-rsa-broken-2030/.

[Jaq25] S. Jaques. "Quantum landscape – 2025 update", Accessed: May 28, 2026. [Online]. Available: https://sam-jaques.appspot.com/quantum_landscape.

[JD24] S. Jendral and E. Dubrova, *Single-trace side-channel attacks on MAYO exploiting leaky modular multiplication*, Cryptology ePrint Archive, Paper 2024/1850, 2024. DOI: 10.1145/3733820.3764682. [Online]. Available: https://eprint.iacr.org/2024/1850.

[KNK+25] S. Kundu, Q. Norga, et al., "mUOV: Masking the unbalanced oil and vinegar digital signature scheme at first- and higher-order", in *ACM CCS 2025: 32nd Conference on Computer and Communications Security*, C.-Y. Huang, J.-C. Chen, et al., Eds., Taipei, Taiwan: ACM Press, Oct. 2025, pp. 1994–2008. DOI: 10.1145/3719027.3765188.

[KPG99] A. Kipnis, J. Patarin, and L. Goubin, "Unbalanced Oil and Vinegar signature schemes", in *Advances in Cryptology – EUROCRYPT'99*, J. Stern, Ed., ser. Lecture Notes in Computer Science, vol. 1592, Prague, Czech Republic: Springer Berlin Heidelberg, Germany, May 1999, pp. 206–222. DOI: 10.1007/3-540-48910-X_15.

[KS98] A. Kipnis and A. Shamir, "Cryptanalysis of the Oil & Vinegar Signature Scheme", in *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, ser. Lecture Notes in Computer Science, vol. 1462, Springer, 1998, pp. 257–266. DOI: 10.1007/BFb0055733.

[MAY] MAYO Team. "MAYO: About", Accessed: Jun. 12, 2026. [Online]. Available: https://pqmayo.org/about/.

[Mic] Microchip Technology Inc. "Atmel SAM4S Series: 32-bit Cortex-M4 Microcontroller Datasheet", Accessed: Apr. 20, 2026. [Online]. Available: https://www1.microchip.com/downloads/en/DeviceDoc/Atmel-11100-32-bit%20Cortex-M4-Microcontroller-SAM4S_Datasheet.pdf.

[Nan20] G. Nannicini, "An introduction to quantum computing, without the physics", *SIAM Review*, vol. 62, no. 4, pp. 936–981, 2020. DOI: 10.1137/18M1170650.

[Nas25] Nasjonal sikkerhetsmyndighet (NSM). "Kvantemigrasjon – nye, kvantesikre standarder", Accessed: May 28, 2026. [Online]. Available: https://nsm.no/fagomrader/digital-sikkerhet/kryptosikkerhet/kvantemigrasjon/.

[Nat15] National Institute of Standards and Technology, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", U.S. Department of Commerce, Federal Information Processing Standards Publication FIPS PUB 202, Aug. 2015. DOI: 10.6028/NIST.FIPS.202.

[Nat16] National Institute of Standards and Technology. "NIST Asks Public to Help Future-Proof Electronic Information", Accessed: May 28, 2026. [Online]. Available: https://www.nist.gov/news-events/news/2016/12/nist-asks-public-help-future-proof-electronic-information.

[Nat17] National Institute of Standards and Technology. "Post-Quantum Cryptography". Updated December 11, 2025, Accessed: May 28, 2026. [Online]. Available: https://csrc.nist.gov/Projects/post-quantum-cryptography.

[Nat22] National Institute of Standards and Technology (NIST). "Request for additional digital signature schemes for the post-quantum cryptography standardization process", Accessed: May 29, 2026. [Online]. Available: https://csrc.nist.gov/News/2022/request-additional-pqc-digital-signature-schemes.

[Nat23] National Institute of Standards and Technology, "Digital Signature Standard (DSS)", National Institute of Standards and Technology, Federal Information Processing Standards Publication NIST FIPS 186-5, Feb. 2023. DOI: 10.6028/NIST.FIPS.186-5.

[Nat24a] National Institute of Standards and Technology, "Module-Lattice-Based Digital Signature Standard", National Institute of Standards and Technology, Federal Information Processing Standards Publication NIST FIPS 204, Aug. 2024. DOI: 10.6028/NIST.FIPS.204.

[Nat24b] National Institute of Standards and Technology. "NIST Releases First 3 Finalized Post-Quantum Encryption Standards", Accessed: May 29, 2026. [Online]. Available: https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards.

[Nat24c] National Institute of Standards and Technology, "Stateless Hash-Based Digital Signature Standard", National Institute of Standards and Technology, Federal Information Processing Standards Publication NIST FIPS 205, Aug. 2024. DOI: 10.6028/NIST.FIPS.205.

[Nat25] National Institute of Standards and Technology (NIST), *"trapdoor" - glossary of key information security terms*, 2025. Accessed: May 10, 2026. [Online]. Available: https://csrc.nist.gov/glossary/term/trap_door.

[Nat26] National Institute of Standards and Technology. "Post-Quantum Cryptography: Additional Digital Signature Schemes – Round 3 Additional Signatures". Updated May 14, Accessed: May 27, 2026. [Online]. Available: https://csrc.nist.gov/projects/pqc-dig-sig/round-3-additional-signatures.

[Newa] NewAE. "ChipWhisperer-Husky", ChipWhisperer Documentation, Accessed: Apr. 11, 2026. [Online]. Available: https://chipwhisperer.readthedocs.io/en/latest/Capture/ChipWhisperer-Husky.html.

[Newb] NewAE. "Custom Firmware on ChipWhisperer Targets", ChipWhisperer Documentation, Accessed: Apr. 11, 2026. [Online]. Available: https://chipwhisperer.readthedocs.io/en/5.7.0/custom-firmware.html.

[Newc] NewAE. "CW308T-STM32F", ChipWhisperer Documentation, Accessed: Apr. 11, 2026. [Online]. Available: https://chipwhisperer.readthedocs.io/en/latest/chipwhisperer-target-cw308t/CW308T_STM32F/README.html.

[Newd] NewAE. "CW312T-SAM4S", ChipWhisperer Documentation, Accessed: Apr. 11, 2026. [Online]. Available: https://chipwhisperer.readthedocs.io/en/latest/chipwhisperer-target-cw308t/CW312T_ATSAM4S/README.html.

[Newe] NewAE. "How long is the target operation?", ChipWhisperer Documentation, Accessed: Apr. 21, 2026. [Online]. Available: https://chipwhisperer.readthedocs.io/en/latest/cw_tips_tricks/basics/trig_count.html.

[Newf] NewAE. “How to use streaming mode?”, ChipWhisperer Documentation, Accessed: Apr. 11, 2026. [Online]. Available: https://chipwhisperer.readthedocs.io/en/latest/cw_tips_tricks/basics/stream.html.

[Newg] NewAE. “Overview”, ChipWhisperer Documentation, Accessed: Apr. 11, 2026. [Online]. Available: https://chipwhisperer.readthedocs.io/en/latest/getting-started.html.

[Newh] NewAE. “Scope API”, ChipWhisperer Documentation, Accessed: Apr. 21, 2026. [Online]. Available: https://chipwhisperer.readthedocs.io/en/latest/scope-api.html.

[Newi] NewAE. “SimpleSerial Documentation”. version 5.7.0, ChipWhisperer Documentation, Accessed: Apr. 11, 2026. [Online]. Available: https://chipwhisperer.readthedocs.io/en/5.7.0/simpleserial.html.

[Newj] NewAE. “Simpleserial Documentation”, ChipWhisperer Documentation, Accessed: Apr. 11, 2026. [Online]. Available: https://chipwhisperer.readthedocs.io/en/latest/simpleserial.html.

[New25] NewAE Technology Inc. “ChipWhisperer Husky & HuskyPlus: Care & Feeding”, Accessed: Apr. 11, 2026. [Online]. Available: https://media.newae.com/manuals/cwhusky_cwhuskyplus_manual_may2025.pdf.

[New26] NewAE Technology Inc. “ChipWhisperer GitHub repository”. commit: 598e784, NewAE Technology Inc., Accessed: Apr. 13, 2026. [Online]. Available: https://github.com/newaetech/chipwhisperer.

[OG21] M. Ouladj and S. Guilley, *Side-Channel Analysis of Embedded Systems: An Efficient Algorithmic Approach*, 1st ed. Cham: Springer Cham, 2021, p. 153, ISBN: 978-3-030-77222-2. DOI: 10.1007/978-3-030-77222-2.

[OSSS97] A. Odlyzko, C. P. Schnorr, et al., “Cryptography (Dagstuhl Seminar 9739)”, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, Dagstuhl Seminar Report 190, 1997, pp. 1–15. DOI: 10.4230/DagSemRep.190.

[Pea01] K. Pearson, “Liii. on lines and planes of closest fit to systems of points in space”, *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901. DOI: 10.1080/14786440109462720.

[PPG24] C. Paar, J. Pelzl, and T. Güneysu, *Understanding Cryptography: From Established Symmetric and Asymmetric Ciphers to Post-Quantum Algorithms*, 2nd ed. Berlin, Heidelberg: Springer, 2024, ISBN: 978-3-662-69007-9. DOI: 10.1007/978-3-662-69007-9.

[PQC25] PQCMayo. “MAYO-M4: Arm Cortex-M4 implementation of MAYO”. commit: 29b1436, Accessed: Apr. 11, 2026. [Online]. Available: https://github.com/PQCMayo/MAYO-M4.

[PQC26] PQCMayo. “MAYO-C: C implementation of mayo”, Accessed: Apr. 11, 2026. [Online]. Available: https://github.com/PQCMayo/MAYO-C.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978, ISSN: 0001-0782. DOI: 10.1145/359340.359342.

[Scr25] A. Scrivano, *A comparative study of classical and post-quantum cryptographic algorithms in the era of quantum computing*, 2025. [Online]. Available: https://arxiv.org/abs/2508.00832.

[Sho97] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer", *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, Oct. 1997, ISSN: 1095-7111. DOI: 10.1137/s0097539795293172.

[Sma16] N. P. Smart, *Cryptography Made Simple* (Information Security and Cryptography). Springer, 2016, ISBN: 978-3-319-21935-6. DOI: 10.1007/978-3-319-21936-3.

[STM] STMicroelectronics. "PM0214: STM32 Cortex-M4 MCUs and MPUs Programming Manual", Accessed: Apr. 20, 2026. [Online]. Available: https://www.st.com/resource/en/programming_manual/pm0214-stm32-cortexm4-mcus-and-mpus-programming-manual-stmicroelectronics.pdf.

[Uni26a] United Nations Department of Economic and Social Affairs. "Goal 16: Promote peaceful and inclusive societies for sustainable development, provide access to justice for all and build effective, accountable and inclusive institutions at all levels", United Nations, Accessed: May 29, 2026. [Online]. Available: https://sdgs.un.org/goals/goal16.

[Uni26b] United Nations Department of Economic and Social Affairs. "Goal 9: Build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation", United Nations, Accessed: May 29, 2026. [Online]. Available: https://sdgs.un.org/goals/goal9.

[Uni26c] United Nations Department of Economic and Social Affairs. "The 17 Goals", United Nations, Accessed: May 29, 2026. [Online]. Available: https://sdgs.un.org/goals.

[VGP+26] N. Vishwaajith, A. Ganguly, et al., *SCA-MQDSA: Side-channel analysis of multivariate digital signature implementations*, Cryptology ePrint Archive, Paper 2026/228, Version: 20260211:122321, 2026. [Online]. Available: https://eprint.iacr.org/2026/228.

[Vik25] N. M. Viken, "Towards Understanding Side-Channel Security in Post-Quantum Digital Signatures", Department of Information Security and Communication Technology, NTNU – Norwegian University of Science and Technology, Project report in TTM4502, Dec. 2025.

[VO22] J. Van Woudenberg and C. O'Flynn, *The Hardware Hacking Handbook: Breaking Embedded Security with Hardware Attacks*. 245 8th Street, San Francisco, CA 94103: No Starch Press, 2022, ISBN: 978-1-59327-874-8.

[WWK+24] D. Wagner, N. Weaver, et al. "Computer security", Accessed: Apr. 29, 2026. [Online]. Available: https://textbook.cs161.org/crypto/signatures.html.

[ZF05] Y. Zhou and D. Feng, *Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing*, Cryptology ePrint Archive, Report 2005/388, 2005. Accessed: May 28, 2025. [Online]. Available: https://eprint.iacr.org/2005/388.

# Appendix A
# Recover Oil Vector

**Listing A.1:** Function used to update the simulated accumulator during vector recovery.

```
def update_acc_for_coeff(acc_state, guessed_coeff, z, k0, attack_P1, v,
    limbs):
    for c in range(v):
        if c == z:
            continue

        r0 = min(z, c)
        c0 = max(z, c)
        p1_block = get_P1_block(attack_P1, r0, c0, limbs, v)

        acc_before = acc_state[c, k0, :].copy()
        acc_after = m_vec_mul_add_model(
                    p1_block,
                    guessed_coeff,
                    acc_before)
        acc_state[c, k0, :] = acc_after

    return acc_state
```

**Listing A.2:** Accumulator initialization, where `attack_P2` is the public matrix **P2**.

```
acc_state_init = np.array(attack_P2, dtype=np.uint64).reshape(v, o, limbs).
    copy()
```

**Listing A.3:** Simplified function used to recover an oil vector using a lookahead mechanism. Validation and debugging details are omitted for readability.

```
def recover_oil_vector(column, segment_records, acc_state,
                    attack_P1, v, o, limbs, poi_list, num_rows,
                    ambiguity_margin, lookahead_top_k, target_len,):

    recovered_vector = []

    # Single coefficient recovery for one row and
    # compute margin between best and second-best CPA candidates
    def analyze_row(acc_state_local, row_idx):
        guess, corr = recover_oil(
            segment_records=segment_records,
            acc_state=acc_state_local,
            attack_P1=attack_P1,
            v=v,
            o=o,
            limbs=limbs,
            z=row_idx,
            k=column,
            poi_list=poi_list,
            target_len=target_len,)

        abs_corr = np.abs(corr)
        masked = np.array(abs_corr, copy=True)
        masked[guess] = -np.inf
        next_best_guess = int(np.argmax(masked))
        margin = abs_corr[guess] - abs_corr[next_best_guess]
        return guess, corr, margin

    for row in range(num_rows):
        # recover the current coefficient using ordinary CPA
        guess, corr, margin = analyze_row(acc_state, row)
        final_guess = guess

        # If the best candidate is not clearly separated from the
        # second-best candidate, use lookahead
        if row + 1 < num_rows and margin < ambiguity_margin:
            abs_corr = np.abs(corr)

            # Only evaluate the strongest candidates from the current row.
            top_candidates = np.argsort(abs_corr)[-lookahead_top_k:][::-1]
            best_score = -np.inf
            best_candidate = None
```

```
            for cand in top_candidates:
                cand = int(cand)
                masked = np.array(abs_corr, copy=True)
                masked[cand] = -np.inf
                next_best = int(np.argmax(masked))
                current_margin = abs_corr[cand] - abs_corr[next_best]

                # Temporarily update the accumulator state as if this
                # candidate were the correct coefficient.
                candidate_acc_state = update_acc_for_coeff(
                    acc_state=acc_state.copy(), guessed_coeff=cand,
                    attack_P1=attack_P1, v=v,
                    limbs=limbs, z=row, k0=column,)

                # Test how well the resulting accumulator state explains
                # the leakage for the next coefficient.
                next_guess, next_corr, next_margin = analyze_row(
                candidate_acc_state, row + 1,)

                # Prefer candidates that give both strong current evidence
                # and a plausible continuation in the next row.
                score = (
                    abs_corr[cand] + current_margin
                    + np.abs(next_corr[next_guess]) + next_margin)

                if score > best_score:
                    best_score = score
                    best_candidate = cand

            final_guess = best_candidate

        recovered_vector.append(final_guess)

        # Permanently update the accumulator state with the selected
        # coefficient before moving to the next row.
        acc_state = update_acc_for_coeff(
            acc_state=acc_state,
            guessed_coeff=final_guess,
            attack_P1=attack_P1,
            v=v,
            limbs=limbs,
            z=row,
            k0=column,)

    return recovered_vector, acc_state
```

# Appendix B
# Recover Oil Matrix

**Listing B.1:** Function used to recover the entire oil matrix by applying `recover_oil_vector` to each row separately.

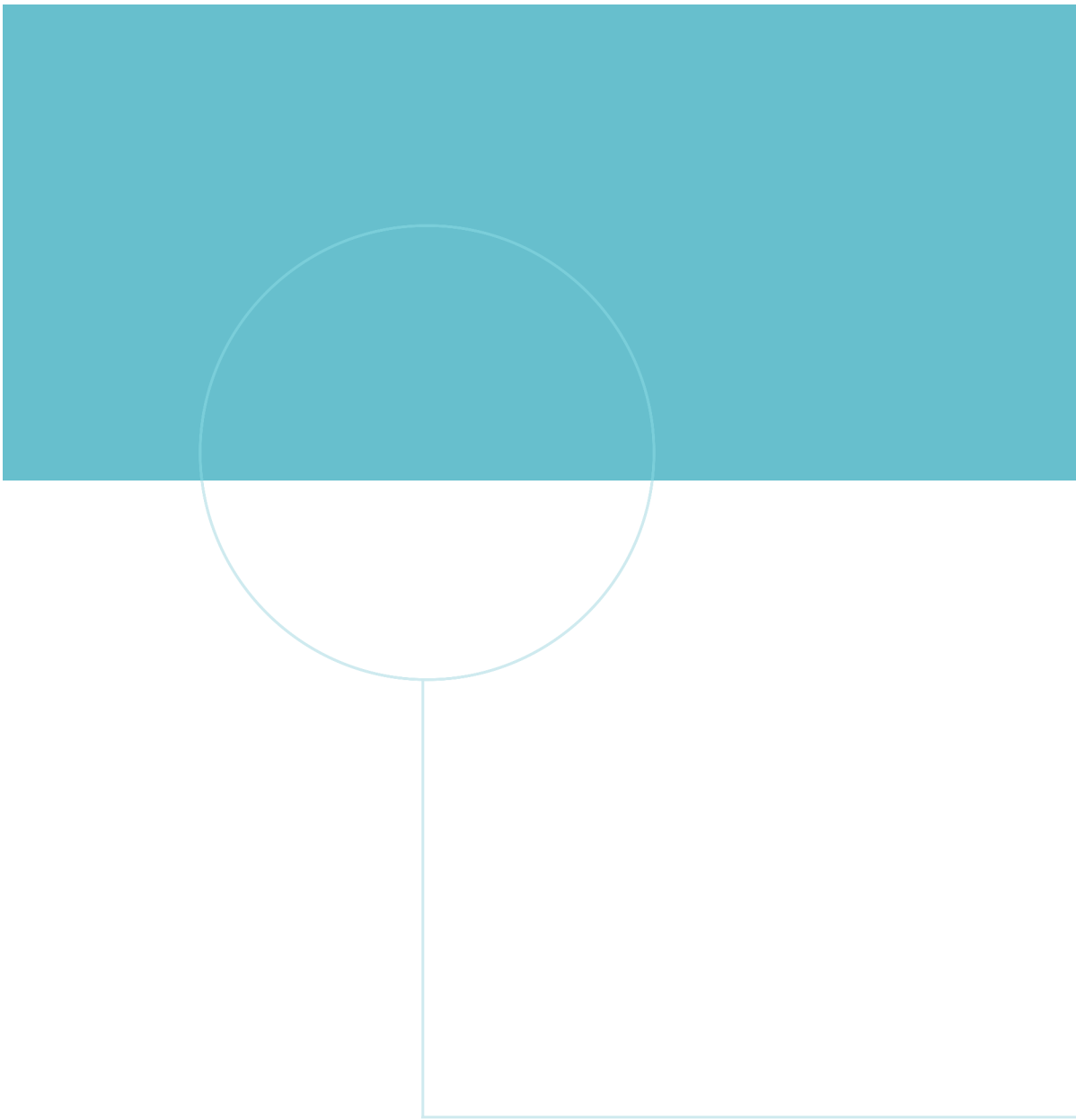
```
def recover_oil_matrix(segment_records, acc_state_init, attack_P1,
    v, o, limbs, poi_list=None, num_rows=None, ambiguity_margin=0.25,
    lookahead_top_k=3, target_len=305,):

    acc_state = np.array(acc_state_init, copy=True)
    recovered_O_matrix = np.zeros((num_rows, o), dtype=np.int64)

    for column in range(o):
        oil_vector, acc_state = recover_oil_vector(
            column=column,
            segment_records=segment_records,
            acc_state=acc_state,
            attack_P1=attack_P1,
            v=v,
            o=o,
            limbs=limbs,
            poi_list=poi_list,
            num_rows=num_rows,
            ambiguity_margin=ambiguity_margin,
            lookahead_top_k=lookahead_top_k,
            target_len=target_len)

        recovered_O_matrix[:, column] = np.asarray(oil_vector, dtype=np.
    int64)

    return recovered_O_matrix, acc_state
```



NTNU

Norwegian University of
Science and Technology