EFFICIENT AND NON-PROFILED SIDE-CHANNEL ATTACKS AGAINST POST-QUANTUM CRYPTOGRAPHY

by
TOLUN TOSUN

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfilment of
the requirements for the degree of Doctor of Philosophy

Sabancı University
July 2025

EFFICIENT AND NON-PROFILED SIDE-CHANNEL ATTACKS AGAINST POST-QUANTUM CRYPTOGRAPHY

Approved by:

Prof. ERKAY SAVAŞ .....
(Dissertation Supervisor)

Prof. ALBERT LEVİ .....

Assoc. Prof. KAMER KAYA .....

Prof. BERNA ÖRS YALÇIN .....

Assoc. Prof. CİHANGİR TEZCAN .....

Date of Approval: July 23, 2025

ABSTRACT

EFFICIENT AND NON-PROFILED SIDE-CHANNEL ATTACKS AGAINST POST-QUANTUM CRYPTOGRAPHY

TOLUN TOSUN

Computer Science and Engineering, Ph.D. Dissertation, July 2025

Dissertation Supervisor: Prof. Erkay Savaş

Keywords: lattice-based cryptography, side-channel analysis, correlation power analysis, kyber, dilithium

This dissertation explores side-channel attacks targeting Post-Quantum Cryptography (PQC). The focus is on lattice-based PQC algorithms standardized by NIST during the course of this research: the Dilithium digital signature scheme (ML-DSA) and the Kyber key encapsulation mechanism (ML-KEM). Both unprotected and protected implementations of these algorithms are considered. The study particularly focuses on the non-profiled class of attacks, which does not rely on access to a clone of the target device.

Existing non-profiled attack methodologies are revisited and improved, particularly in terms of the required number of traces and also the attack run-time. While both aspects are addressed, the trace complexity is given greater emphasis. Regarding the attack run-time efficiency, an attack methodology applicable to Kyber and specific implementations of Dilithium is introduced, achieving speedups of up to three orders of magnitude.

The thesis explores the application of higher-order non-profiled attacks to Lattice-Based Cryptography (LBC). These attacks face unique challenges due to the so-called arithmetic masking schemes employed in protected LBC implementations. These challenges are analyzed in depth, and novel solutions are proposed. Performing higher-order non-profiled attacks require to compute so-called the optimal prediction function. This work presents efficient methods for deriving these functions in the context of arithmetic masking and LBC, including explicit formulas in

specific cases—namely, when modular reduction is performed in a signed fashion centered around zero.

Importantly, the thesis demonstrates that using signed arithmetic introduces a significant vulnerability by creating a strong dependency between the signs of intermediate variables and the observed leakage.

Experimental results are presented for both simulated and real-device settings, covering implementations from unprotected up to third-order masked. These results are unique in the literature, representing the first demonstration of non-profiled attacks against higher-order masked implementations of LBC. The findings reveal that non-profiled side-channel attacks pose a serious threat to masked implementations. For example, third-order masked implementations of Dilithium and Kyber are successfully attacked with only 2400 and 14500 traces, respectively.

Furthermore, the thesis addresses the scenario in which the attacker does not know the leakage function of the device. A novel two-step attack combining generic SCA distinguishers with lattice reduction techniques is proposed. Experimental results show that this approach enables successful non-profiled attacks even when the victim implementation employs masking protection and the device's leakage characteristics are unknown.

ÖZET

KUANTUM-SONRASI KRİPTOGRAFİYE KARŞI VERİMLİ VE PROFİL GEREKTİRMEYEN YAN KANAL SALDIRILARI

TOLUN TOSUN

Bilgisayar Bilimi ve Mühendisliği, Doktora Tezi, Temmuz 2025

Tez Danışmanı: Prof. Dr. Erkay Savaş

Anahtar Kelimeler: kafes-tabanlı kriptografi, yan kanal analizi, korelasyon güç analizi, kyber, dilithium

Bu tez, Kuantum Sonrası Kriptografi'yi (PQC) hedef alan yan kanal saldırılarını incelemektedir. Çalışmanın odağında, bu araştırma süresince NIST tarafından standardize edilen kafes tabanlı PQC algoritmaları yer almaktadır: Dilithium dijital imzalama şeması (ML-DSA) ve Kyber anahtar kapsülleme mekanizması (ML-KEM). Bu algoritmaların hem korumasız hem de yan-kanal saldırılarına karşı korumalı gerçeklemeleri ele alınmıştır. Çalışma, özellikle hedef cihazın bir kopyasına erişim olmadan gerçekleştirilebilen, profil gerektirmeyen saldırı sınıfına odaklanmaktadır.

Mevcut profil gerektirmeyen saldırı yöntemleri yeniden ele alınmış ve bunlar özellikle ihtiyaç duyulan elektriksel güç izi sayısı ve saldırı çalışma süresi açısından iyileştirilmiştir. Çalışmalarımızda her iki parametre iyileştirilmeye çalışılsa da, güç izi karmaşıklığına daha fazla öncelik verilmiştir. Saldırı çalışma süresi açısından ise, Kyber ve Dilithium'un belirli gerçeklemelerine uygulanabilen bir saldırı yöntemi sunulmuş ve bu sayede çalışma süresinde bin kata kadar hız artışı sağlanmıştır.

Tez, yüksek dereceden, profil gerektirmeyen saldırıların kafes tabanlı kriptografi (LBC) üzerindeki uygulanabilirliğini de incelemektedir. Bu tür saldırılar, korumalı LBC gerçeklemelerinde kullanılan aritmetik maskeleme şemaları olarak bilinen koruma önlemleri nedeniyle özgün zorluklarla karşı karşıyadır. Bu zorluklar detaylı olarak analiz edilmiş ve özgün çözümler önerilmiştir. Yüksek dereceden profil gerektirmeyen saldırıların gerçekleştirilmesi, "eniyilenmiş tahmin fonksiyonu" olarak bilinen fonksiyonun hesaplanmasını gerektirir. Bu çalışma, aritmetik maskeleme ve

LBC bağlamında bu fonksiyonların elde edilmesine yönelik verimli yöntemler sunmakta; özellikle, modüler indirgeme işleminin sıfır etrafında merkezlenmiş işaretli biçimde yapıldığı özel durumlar için açık formüller sağlamaktadır.

Yine önemli bir katkı olarak tezde, işaretli aritmetik kullanımının, ara değişkenlerin işareti ile gözlemlenen sızıntı arasında güçlü bir bağımlılık oluşturarak ciddi bir güvenlik açığına neden olduğu gösterilmektedir.

Hem simülasyon ortamında hem de gerçek cihazlarda gerçekleştirilen deneysel sonuçlar, korumasız gerçeklemelerden üçüncü dereceden maskeleme kullanan gerçeklemelere kadar geniş bir yelpazeyi kapsamaktadır. Bu sonuçlar, literatürde bir ilk olup, yüksek dereceden maskeleme içeren LBC uygulamalarına karşı gerçekleştirilen profil gerektirmeyen saldırıların ilk başarılı gösterimini sunmaktadır. Bulgular, profil gerektirmeyen yan kanal saldırılarının maskeleme kullanan gerçeklemeler için ciddi bir tehdit oluşturduğunu ortaya koymaktadır. Örneğin, üçüncü dereceden maskeleme kullanılan Dilithium ve Kyber gerçeklemelerine karşı yan-kanal saldırılarının, sırasıyla, yalnızca 2400 ve 14500 güç izi kullanılarak, başarıyla uygulanabildiği gösterilmiştir.

Ayrıca tez, saldırganın cihazın sızıntı fonksiyonunu bilmediği senaryoyu da ele almaktadır. Bu bağlamda, genel yan-kanal analizi (SCA) ayrıştırıcıları ile kafes indirgeme tekniklerini birleştiren özgün, iki aşamalı bir saldırı yöntemi önerilmiştir. Deneysel sonuçlar, bu yaklaşımın maskeleme koruması içeren ve sızıntı fonksiyonu bilinmeyen durumlarda dahi başarılı, profil gerektirmeyen saldırılar yapılmasını mümkün kıldığını göstermektedir.

ACKNOWLEDGEMENTS

First of all, I would like to congratulate myself for having the courage, patience, and perseverance to complete this thesis. Without such dedication and hard work, this outcome would not have been possible. I am also deeply grateful for the endless support of my family throughout this journey.

I am thankful to my advisor Prof. Erkay Savaş for his invaluable support, trust and guidance throughout my PhD journey. I am also grateful for the opportunity to work and co-author with some of the best researchers in the field of side-channel analysis, Prof. Amir Moradi and Prof. Elisabeth Oswald. I am deeply thankful to them for everything I have learned from their guidance and expertise.

I would like to acknowledge Analog Devices Istanbul Office, where I worked during the first five years of my PhD. Working there allowed me to learn side-channel analysis in a professional setting and to perform measurements in a fully equipped laboratory. This experience significantly contributed to my progress in this field.

The work described in this thesis has been supported in part by European Union through the Twinning Project 101079319 (acronym enCRYPTON). Views and opinions expressed are those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

To the all side-channels that betrayed their secrets,

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATONS

CPA Correlation Power Analysis . . xvii, xviii, 2, 4, 37, 48, 50, 51, 89, 98, 103, 104

CRT Chinese Remainder Theorem 24

DPA Differential Power Analysis 2

FFT Fast Fourier Transform 24

HD Hamming Distance 2

HOCPA Higher-Order Correlation Power Analysis . . xv, xvii, xviii, xix, 4, 50, 51, 79, 96, 97, 99, 101, 103, 104, 105, 110, 123, 137

HW Hamming Weight . . . . . xv, xvii, xviii, 2, 5, 82, 83, 84, 85, 91, 101

KEM Key Encapsulation Mechanism 1, 9

KW Kruskal–Wallis . . . . . xvi, 3, 43, 53, 54, 131, 133, 137

LBC Lattice-Based Cryptography . iv, v, 3, 4, 5, 6, 8, 9, 16, 29, 30, 31, 48, 55, 82, 84, 85, 97, 127, 136, 137

MI Mutual Information 39, 88

M-LWE Module Learning With Errors 1

MIA Mutual Information Analysis . . . xvi, 3, 39, 40, 53, 88, 89, 128, 131, 133, 137

ML-KEM Key Encapsulation Mechanism based on Module Lattices . . . . . iv, 1

ML-DSA Digital Signature Algorithm based on Module Lattices . . . . . iv, 1

MMIA Multivariate Mutual Information Analysis . . . . . xv, xvii, 52, 53, 79, 81

NTT Number Theoretic Transform xi, 3, 4, 6, 8, 11, 24, 25, 26, 31, 55, 56, 57, 58, 59, 61, 62, 95, 96, 99, 102, 104, 136

PQC Post-Quantum Cryptography . . . . . iv, 1, 16, 62, 136

R-LWE Ring Learning With Errors 1

SCA Side-Channel Analysis v, xiv, xvii, xix, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 32, 33, 34, 35, 47, 50, 55, 56, 57, 62, 63, 82, 83, 86, 97, 99, 105, 127, 131, 134, 136, 137

SIS Short Integer Solution 1

SNR Signal-to-noise Ratio ..... xix, 134, 135

1. INTRODUCTION

Security of public-key cryptosystems relies on the hardness of well-known mathematical problems such as the discrete logarithm problem for the elliptic curve cryptography (ECC) (Koblitz, 1987; Miller, 1986) and the digital signature algorithm (DSA) (Schnorr, 1990) or the integer factorization problem for RSA (Rivest, Shamir & Adleman, 1978). While those hard problems are conjectured to be secure against known cryptanalytic algorithms running on classical computers, it has been shown that Shor's algorithm (Shor, 1994) can solve them in polynomial time on a large-scale quantum computer.

To address the quantum threat, the National Institute of Standards and Technology (NIST) has announced the standardization process for the public-key PQC algorithms in 2016. The standardization process covers quantum-resistant digital signature schemes, and public-key encryption and key-establishment algorithms. Currently, the contest is at the fourth round with already standardized algorithms. Lattice-based schemes, based on various hard lattice problems, facilitate the construction of quantum resilient public-key cryptography with a promising level of efficiency. Among the winners, the lattice-based digital signature algorithm Crystals-Dilithium Ducas, Kiltz, Lepoint, Lyubashevsky, Schwabe, Seiler & Stehlé (2018) is based on the M-LWE (Langlois & Stehlé, 2015), and Module-SIS problems, while Crystals-Kyber Bos, Ducas, Kiltz, Lepoint, Lyubashevsky, Schanck, Schwabe, Seiler & Stehlé (2018) is M-LWE based KEM. M-LWE is the module version of the R-LWE problem (Regev, 2009). As Kyber and Dilithium are members of the same family, Crystals, they have several building blocks in common. These algorithms are standardized under the names ML-KEM and ML-DSA.

In cryptanalysis, SCA attacks are the ones that target the weaknesses in implementations rather than algorithm specifications, by collecting side information such as running time or power consumption that can leak sensitive (intermediate) information during the execution of the targeted cryptographic operation. Side-channel attacks are considered as one of the main threats, particularly for embedded devices because of the simplicity of side-information collection, such as IoT chips, which

sign sensor data before transmission. Timing attacks are the first and simplest type of side-channel attacks in the history of cryptography, presented by Kocher (1996), which makes use of the fact that the execution time of an algorithm can reveal sensitive information regarding the secret data; mostly occurred due to performance optimizations such as branching, conditional statements, cache hits/misses. In a few years after this breakthrough, The first power analysis-based attacks, namely Simple Power Analysis (SPA), and DPA were reported by Kocher, Jaffe & Jun (1999), which take advantage of the fact that the power consumption of a device is related to the running application and the processed data (Mangard, Oswald & Popp, 2008). The CPA is proposed by Brier, Clavier & Olivier (2004), which models the power consumption of the device under test and measures the correlation of the model with real-world data to test secret value hypotheses. The power leakage of the device/implementation is often modeled with the HW of the intermediate data or HD between the intermediate data. The Electromagnetic (EM) side-channels Agrawal, Archambeault, Rao & Rohatgi (2002) are similar to power analysis as any attack suit designed for power leakage can be practiced with EM leakage while it can supply more precise information about the sensitive intermediate data. Masking is one of the most promising countermeasures against power/EM attacks, which randomizes the intermediate data with secret sharing, so that characteristics of sensitive data are not reflected in power consumption.

The introduction of profiled attacks (also known as template attacks) by Chari, Rao & Rohatgi (2002) marked the emergence of a new category of power-based SCA attacks. These attacks assume access to an identical clone of the victim device for power measurement and modeling, referred to as the profiling device. The profiling device must be configurable by the attacker, *i.e.*, the attacker must be able to set arbitrary secret keys on it. By experimenting on this device, the attacker characterizes the leakage model of the victim device. The resulting model can then be applied to side-channel data collected from the actual victim device to recover the secret key. In contrast, CPA and DPA are essentially non-profiled SCA attacks, which do not require a profiling device. The attacker relies on generic statistical methods to recover the secret key from the side-channel data.

Although the Hamming weight assumption usually holds for software implementations, the leakage characteristic of the victim device can be more complex for hardware implementations (Gao, Marshall, Page & Oswald, 2020; Levi, Bellizia & Standaert, 2019). In such cases, and especially when the leakage model cannot be recovered through a profiling device, it remains unknown to the attacker. A common approach to perform non-profiled attacks in this scenario is to use so-called generic distinguishers, which perform the profiling on-the-fly. Efficient examples of generic

distinguishers are MIA (Gierlichs, Batina, Tuyls & Preneel, 2008) and KW (Yan, Oswald & Roy, 2023).

JIL Rating (Librabry, 2020) is a widely used metric to assess the complexity of side-channel attacks; the higher the JIL score, the harder to perform the attack. As the time needed to apply the attack is a factor in the overall rating, both the number of traces and the attack run-time affect the rating of the attack, constituting an important motivation for this work.

While lattice-based cryptographic algorithms are resistant to quantum attacks, protecting them against SCA attacks is a highly active research area, and potential attack scenarios must be carefully considered. Since post-quantum algorithms are intended to replace the existing public-key standards soon and the usage of public-key cryptography in embedded devices will be potentially more extensive. For example, a secure firmware update on an embedded device relies on the security of the employed digital signature while embedded devices are open to timing, and power/EM attacks by nature. With increasing interest, several attacks and countermeasures have been proposed for PQC candidates in the literature. The majority of the existing literature focuses on profiled attacks, with examples including Dubrova, Ngo, Gärtner & Wang (2023); Primas, Pessl & Mangard (2017); Ravi, Roy, Chattopadhyay & Bhasin (2020). However, non-profiled attacks are also an important area of study, as they offer a lower-cost attack path. Moreover, a profiling device may not be available to the attacker in many scenarios. For example, when the target device is a smart card issued by an authority, the attacker is unlikely to have access to an identical open clone for profiling.

Polynomial multiplication is the core operation for practical constructions of LBC, which is efficiently implemented using the NTT algorithm. This operation is also the main target for non-profiled SCA attacks against LBC (Chen, Karabulut, Aysu, Ma & Jing, 2021; Kuo & Takayasu, 2023; Mujdei, Wouters, Karmakar, Beckers, Bermudo Mera & Verbauwhede, 2024; Qiao, Liu, Zhou, Shao & Sun, 2023; Rodriguez, Bruguier, Valea & Benoit, 2023; Steffen, Land, Kogelheide & Güneysu, 2023). This is expected, as polynomial multiplication involves both secret and public data as input operands, which is a necessary condition for non-profiled attacks. A technique referred as *incomplete NTT* is introduced to handle rings of special structures as well as to improve efficiency Lyubashevsky & Seiler (2019a). Kyber employs an incomplete NTT due to its parameter selection, whereas Dilithium's parameters allow for a regular NTT, referred to as complete. However, certain implementations of Dilithium adopt an incomplete NTT for efficiency (Abdulrahman, Hwang, Kannwischer & Sprenkels, 2022). The characteristics of SCA attacks

on incomplete-NTT-based implementations differ slightly from those on complete NTT implementations (Mujdei et al., 2024). This study examines both scenarios in detail.

An important building block that significantly impacts the efficiency of the NTT algorithm is the modular reduction of integers, concerning the arithmetic for coefficients of polynomials. Classical techniques such as the Montgomery reduction (Montgomery, 1985) and Barrett reduction (Barrett, 1986) are already applied to LBC by the existing literature (Abdulrahman et al., 2022; Alkim, Bilgin, Cenk & Gérard, 2020; Botros, Kannwischer & Schwabe, 2019; Greconici, Kannwischer & Sprenkels, 2021; Seiler, 2018). The same holds for the relatively new Plantard reduction scheme (Huang, Zhang, Zhao, Liu, Cheung, Koç & Chen, 2022; Plantard, 2021). One important distinction regarding the integer reduction in LBC compared to the RSA and ECC is the bit-length of the number to be reduced. LBC requires a reduction of relatively smaller numbers, that usually fit into a single computer word. For instance, Kyber employs a 12-bit coefficient modulus and Dilithium a 23-bit one. Moreover, the signed representation of integers over a modulus instead of the classical unsigned representation is more desired in LBC (Abdulrahman et al., 2022; Alkim et al., 2020; Botros et al., 2019; Greconici et al., 2021; Huang et al., 2022). That is to make a central reduction (*a.k.a. centered* reduction) to the range $[-q/2, q/2]$ instead of $[0, q)$ for an odd modulus q. The Plantard and Montgomery algorithms enable 2-cycle implementation of central reduction on the ARM Cortex-M4 (Alkim et al., 2020; Greconici et al., 2021; Huang et al., 2022) while Plantard is superior since the output of Montgomery reduction requires an additional subtraction or addition to be correct.

Overall, the central reduction improves the efficiency of LBC implementations; however, it also creates new possibilities for SCA attacks. Respectively, the sign of a number in $[-q/2, q/2]$ becomes the dominant factor influencing its power consumption considering 2's complement to represent negative numbers. Motivated by this, we explore the characteristics of the central reduction in terms of SCA leakage.

To mitigate power based SCA attacks, masking Chari, Jutla, Rao & Rohatgi (1999) is the state-of-the-art countermeasure, and its effectiveness is quantified by the masking order. Performing CPA attacks on masked implementations is referred to as the HOCPA attacks. In this context, a function known as the *optimal prediction function* must be calculated to tackle the masking countermeasure (Prouff, Rivain & Bevan, 2009). While this computation is straightforward for symmetric cryptography, additional effort is required for attacking protected implementations of LBC.

Application of the generic distinguishers in the context of LBC is also challenging

and remains relatively underexplored. This difficulty arises primarily from the fact that the modular multiplications in the NTT based polynomial multiplication are injective functions Heuser, Rioul & Guilley (2014). A known workaround is to use the so-called bit-dropping trick; however, this method incurs information loss, and we demonstrate that its efficacy against masked implementations of LBC remains uncertain.

Table 1.1 Qualitative summary of the attack scenarios covered in this study. The column "Run-Time" indicates improvements in the SCA attack's execution time for the corresponding scenario, while the column "#Traces" denotes exploration and improvements of efficiency in terms of the number of traces required.

(Chapter) Work	**Victim Algorithm**	**Device Leakage Function**	**Protected (Masking)**	**Run-Time**	**#Traces**
(Chapter 5) Tosun & Savas (2024)	Kyber	HW	✓	✓	✗
	Dilithium	HW	✗	✓	✗
(Chapter 6) Tosun et al. (2024)	Kyber	HW	✓	✗	✓
	Dilithium	HW	✓	✗	✓
(Chapter 7) Tosun et al. (2025)	Kyber	HW	✓✓	✗	✓
	Dilithium	HW	✓✓	✗	✓
(Chapter 8) Tosun et al. (2025)	Kyber	Non-HW	✓✓	✓	✓
	Dilithium	Non-HW	✗	✓	✓

✓✓: Higher-Order Masking.

In this work, we study non-profiled SCA attacks on LBC across various scenarios. The attack scenarios, including the victim algorithm, masking protection, device leakage model, and the efficiency consideration addressed, are summarized in Table 1.1, along with related publications and corresponding chapters. The rest of the document is organized as follows. In Chapter 2 we present a brief literature review on the SCA attacks on LBC. In Chapter 3 we provide the necessary background on LBC and SCA. In Chapter 4 we present the outline for the non-profiled SCA attack on NTT-based polynomial multiplication in LBC. In Chapter 5, Chapter 6, Chapter 7, and Chapter 8 we present our contributions in detail. Finally, we conclude the thesis in Chapter 9.

1.1. Main Contributions

The contributions of this thesis are listed as follows:

- To the best of our knowledge, we present the first study in the literature that is particularly developed to effectively exploit the leakage of SCA-protected implementations of LBC without the need for profiling.
- In Chapter 5, we present a novel non-profiled power/EM attack against incomplete NTT-based implementations of polynomial multiplication in LBC, referred to as Zero-Value Filtering Attack (ZV-FA). Our approach improves attack run-time by significantly reducing the number of hypotheses through a filtering technique based on zero-value coefficients in the known input/output polynomials of the operation targeted by the SCA attack.
- In Chapter 6, we show that the central reduction techniques that are widely adapted in LBC lead to a source of effectively exploitable SCA leakage. Particularly, information about the sign of arithmetic shares would ease exploiting the leakage and conducting successful key-recovery attacks.
- In Chapter 6, we show that the employed coefficient modulus as well as the reduction algorithm and the machine word size affect the SCA leakage of masked implementations of LBC, particularly making non-profiled attacks easier to be conducted. For the aforementioned scenarios, we particularly present the so-called optimal correlation and the number of traces required for different noise levels. Our study reveals that SCA attacks on masked implementations with central reduction are significantly more noise-tolerant. In Chapter 7, we extend these results to higher-order masked implementations.
- In Chapter 7, we introduce a novel and efficient method for computing the optimal prediction functions for higher-order SCA attacks using a recursive approach. Our approach enables the derivation of closed-form expressions for higher-order attacks when possible, while also offering advantages in scenarios where closed-form solutions are infeasible, such as when the attacked device uses unsigned modular arithmetic in LBC.
- In Chapter 6 and Chapter 7, we propose novel prediction functions for higher-order SCA attacks against the arithmetic masking in LBC. These functions are particularly effective when the modular arithmetic in the target implementation is signed. The absolute value prediction function introduced in Chapter 6

is effective against first-order masking, whereas the *sin* and *cos* based prediction functions introduced in Chapter 7 are effective for attacking second- and higher-order masking.

- We provide practical SCA attack results against Kyber and Dilithium across various scenarios. Our experiments cover both EM- and power-based SCA attacks using different setups. We target unprotected and protected implementations of Kyber and Dilithium, including first-, second-, and third-order masking. We focus on open-source implementations of those algorithms on the ARM Cortex-M4. Our results are unique in the literature as these are the first higher-order and efficient non-profiled attacks demonstrated against both Kyber and Dilithium. Additionally, we made our attack scripts open-source to ensure reproducibility.

- We experimentally show that only a few hundred traces are required to successfully mount a key-recovery attack on Kyber and Dilithium without profiling, even in many protected scenarios. For example, in our experimental setup, first-order masked implementations of Kyber and Dilithium can be broken using just 250 (Chapter 6) and 190 (Chapter 7) traces, respectively.

- In Chapter 8, we explore application of generic distinguishers, such as MIA and KW, for attacking lattice-based schemes when the leakage model of the victim device is unknown. We present a novel two-step attack on incomplete NTT multiplication (*e.g.* Kyber) by combining generic distinguishers with lattice problems, namely solving instances of the SVP. We show that our approach is particularly favorable against protected implementations.

- In Chapter 8, our study reveals that, recovering the *delta*s, the factors between the even and odd-degree coefficients of Kyber's secret polynomials in incomplete NTT domain, are sufficient to recover the entire secret polynomial.

2. RELATED WORK

There exist many SCA attacks in the literature that target implementations of LBC, which can be categorized into three classes:

- Profiled attacks.
- Non-profiled attacks.
- Public profiled attacks.

The majority of published SCA attacks on LBC are profiled. To this end, several operations of LBC have been selected as the target of these attacks. Among them, (Kim, Lee, Han, Sim & Han, 2020; Primas et al., 2017; Xu, Pemberton, Roy, Oswald, Yao & Zheng, 2021) focus on the NTT transformation. The attack presented by Primas et al. (2017) is distinctive by revealing *single-trace* vulnerabilities of LBC. The authors demonstrate that, under certain conditions, a single measurement from the victim device is sufficient to recover the secret polynomial input to the NTT. Their approach utilizes the belief propagation algorithm to combine leakage from different time points during the victim's NTT computation. It should be emphasized that if the masks during the profiling phase are known (*i.e.* they are also considered in the profiles), single-trace profiling attacks cannot be avoided by masking, since SCA leakage of a single execution is measured, where the masks can also be revealed via the profiles. Bronchain, Azouaoui, ElGhamrawy, Renes & Schneider (2024) presents another profiled attack on NTT-based polynomial multiplication, exploiting the fact that the coefficients of secret polynomials are sampled from a small distribution rather than uniformly from the prime field. The sub-routines targeted by the profiled class is not limited to NTT multiplication. Backlund, Ngo, Gärtner & Dubrova (2023); Dubrova et al. (2023); Wang, Brisfors & Dubrova (2024); Xu et al. (2021) target data conversion primitives between polynomial and binary representations, such as those involving message encoding or decoding. These studies are essentially chosen-ciphertext attacks (CCA) against Kyber, which require sending specially crafted ciphertexts to the decapsulation (i.e., decryption) function of the attacked device. Notably, Dubrova et al. (2023) successfully attacks an open-

source fifth-order masked implementation of Kyber using deep-learning techniques. Another profiled attack targeting binary to polynomial conversion is presented in Marzougui, Ulitzsch, Tibouchi & Seifert (2022), but in the context of Dilithium. Karabulut, Alkim & Aysu (2021) targets the sampling of challenge polynomials in Dilithium, which have a limited number of non-zero coefficients that are also small in magnitude. Ravi et al. (2020) constructs a *plaintext-checking oracle* using side-channel leakage from the so-called FO transform, which is employed in lattice-based KEMs like Kyber to achieve chosen-ciphertext security from chosen-plaintext security. This oracle is then queried with chosen ciphertexts, and by observing variations in the decryptions, the secret polynomial is gradually recovered.

To the best of our knowledge, all non-profiled SCA attacks against LBC target the NTT-based polynomial multiplication Chen et al. (2021); Kuo & Takayasu (2023); Mujdei et al. (2024); Qiao et al. (2023); Rodriguez et al. (2023); Steffen et al. (2023), where the secret polynomial is multiplied with a publicly known polynomial, which corresponds to the ciphertext in Kyber and the challenge (part of the signature) in Dilithium. Mujdei et al. (2024) shows that the performance of non-profiled attacks against the polynomial multiplication directly depends on the employed multiplication algorithm as well as on the parameters such as the polynomial coefficient modulus. While non-profiled attacks on some instances of LBC may require a relatively long time to recover secret polynomials, acceleration is possible in certain scenarios by collecting more measurements, as demonstrated by Chen et al. (2021) against Dilithium. Kuo & Takayasu (2023); Qiao et al. (2023) combines SCA attacks and other cryptanalysis techniques such as lattice attacks. In particular, the authors post-process the output of the SCA attack using lattice attacks, allowing a significant number of incorrect predictions by the SCA to be tolerated and corrected. For example, Kuo & Takayasu (2023) demonstrates that predicting 38 out of 128 secret coefficients in Kyber is sufficient to recover the remaining coefficients through the lattice attack. Besides, it is shown in Rodriguez et al. (2023); Steffen et al. (2023) that the polynomial multiplication can be also effectively targeted in the case of hardware implementations. Except Qiao et al. (2023), the aforementioned non-profiled attacks target unprotected implementations, *i.e.* not masked. On the other hand, all of them assume Hamming Weight-based device leakage and therefore study tools and methods suited for that leakage model.

In addition to the classical profiled and non-profiled attack settings, public profiled attacks are a special case of profiled attacks, in which the profiling step *can be* performed directly on the victim device. As a result, a configurable clone of the victim device is not required. The authors also discuss that their methodology can be considered as a public profiled attack. In this case, the attack still involves

a profiling step, as in traditional profiled attacks. However, the profiling can be performed directly on the target device, eliminating the need for a separate clone device accessible to the attacker. Examples of this class are Backlund et al. (2023); Dubrova et al. (2023); Karabulut et al. (2021); Ravi et al. (2020); Wang et al. (2024); Xu et al. (2021).

The discussed SCA attacks are summarized in Table 2.1. It should be noted that only the relevant target algorithms, Kyber and Dilithium, are listed, even if the original publications included additional algorithms. Furthermore, although many of the listed attacks could, in principle, be extended to masked implementations by leveraging more traces, we only mark those that explicitly demonstrate such an attack. For non-profiled attacks, the "polynomial multiplication" target refers specifically to the element-wise multiplication (*a.k.a.* base multiplication) in the NTT domain. In contrast, for profiled attacks, the "NTT" target refers to the transformation itself, either entering or exiting the NTT domain.

Table 2.1 Qualitative summary of related state-of-the-art SCA attacks.

Work	Class	Post-Process	Chosen-Ciphertexts	Masked	Algorithm	Implementation	Target Function
Mujdei et al. (2024)	NP	✗	✗	✗	K,D*	Cortex-M4✯	poly. mult.
Chen et al. (2021)	NP	✗	✗	✗	K*,D	Ref. C	poly. mult.
Rodriguez et al. (2023)	NP	✗	✗	✗	Dilithium	Hardware	poly. mult.
Steffen et al. (2023)	NP	✗	✗	✗	Dilithium	Hardware	poly. mult.
Qiao et al. (2023)	NP	LA	✗	✓	K,D	✈,Cortex-M4❧,	poly. mult.

Kuo & Takayasu (2023)	NP	LA	✗	✗	K,D*	Cortex-M4★	poly. mult.
Primas et al. (2017)	P	BP	✗	✓	K,D	Cortex-M4✧	NTT
Kim et al. (2020)	P	✗	✗	✗	D	Ref. C	NTT
Kim et al. (2020)	P	✗	✗	✓	D	Ref. C✶	small poly. mult.
Bronchain et al. (2024)	P	BP	✗	✗	Dilithium	✤	small poly.mult
Marzougui et al. (2022)	P	ILP	✗	✗	D	Ref. C	bin. to poly.
Karabulut et al. (2021)✉	PP	✗	✗	✓	D	Ref. C Cortex-M4★	small poly. sampling
Ravi et al. (2020)	PP	✗	✗	✗	K	Cortex-M4★	FO Transform
Wang et al. (2024)	PP	✗	✓	✗	K	Cortex-M4♣	bin. to poly.
Backlund et al. (2023)	PP	✗	✓	✓	K	Cortex-M4❧	poly. to bin.
Xu et al. (2021)	PP	✗	✗	✗	K	Ref. C	NTT
Xu et al. (2021)	PP	✗	✓	✗	K	Cortex-M4★	bin. to poly.
Dubrova et al. (2023)	PP	✗	✓	✓	K	Cortex-M4❧	bin. to poly.

Classes: Non-Profiled (NP), Profiled (P), Public Profiled (PP)

Victim algorithms: Kyber (K), Applies to Kyber but not demonstrated (K*), Dilithium (D), Applies to Dilithium but not demonstrated (D*)

Post-Processing: Lattice Attack (LA), Belief-Propagation (BP), Integer Linear Programming(ILP)

Implementations:

★ from Kannwischer, Rijneveld, Schwabe & Stoffelen (2019), ✧ from Reparaz,

Roy, De Clercq, Vercauteren & Verbauwhede (2016)

❧ from Heinz, Kannwischer, Land, Pöppelmann, Schwabe & Sprenkels (2022), ♣ from Bronchain & Cassiers (2022)

✈ from Azouaoui, Bronchain, Cassiers, Hoffmann, Kuzovkova, Renes, Schönauer, Schneider, Standaert & van Vredendaal (2022)

✶ attacks through an implementation submitted for another project by Alkim, Barreto, Bindel, Krämer, Longa & Ricardini (2020)

✤ not reported

Misc: ✉ challenge polynomial recovery attack

3. BACKGROUND

3.1. Notation

- Matrices and Vectors:
  - Uppercase bold letters denote matrices, while lowercase bold letters denote vectors. Elements are accessed using lower indices in square brackets, such as $\mathbf{A}_{[i,j]}$ for the (i,j)-th entry of a matrix and $\mathbf{a}_{[i]}$ for the i-th entry of a vector. The notation $[a_i]_{i=0}^{m}$ denotes a vector composed of the elements $a_0, a_1, \ldots, a_m$.
- Sets:
  - Uppercase *blackboard bold* letters denote sets. Elements of those are accessed by the lower-index, such as $\mathtt{A}_{[i]}$. We use the notation $\{a_i\}_{i=0}^{m}$ to denote a set containing elements $a_0, a_1, \ldots, a_m$. We exclude standard sets, such as the set of integers $\mathbb{Z}$, from this notation.
- Variables and random variables:
  - Lowercase italic letters denote variables, such as x. Uppercase letters denote random variables, such as X. $\leftarrow$ denotes sampling uniformly at random, such as $X \leftarrow \mathtt{X}$. Upper-index with round brackets denotes samples of random variables, such as $X^{(i)}$.
  - $\mathsf{P}(\cdot)$ denotes the probability function and $\mathsf{E}[\cdot]$ denotes the expected value function.
- Secret shares in masking schemes:
  - Upper-index with curly brackets denotes secret shares of variables used in

masking schemes, such as $X = X^{\{0\}} + X^{\{1\}}$. We also use the shorthand notation $X^{\{0:d-1\}} = \{X^{\{0\}}, X^{\{1\}}, .., X^{\{d-1\}}\}$ to refer to all d shares.

- Polynomials:
  - Polynomials are also denoted by lowercase italic letters, such as $a(x)$. For brevity, we often omit the indeterminate x and simply refer to the polynomial as a. Coefficients of polynomials are accessed by the lower-index with square brackets, such as $a_{[i]}$.
  - "hat" denotes NTT domain representations of polynomials, such as $\hat{a} = \mathsf{NTT}(a)$.
- Functions:
  - *Sans-Serif* letters denote functions, such as F.
  - The function HW_β denotes the Hamming weight computed over a β-bit number represented in 2's complement form.
  - Logarithm is base-2 unless otherwise stated.
- Multiplications:
  - The polynomial multiplication is denoted by $\cdot$, as in $a(x) \cdot b(x)$ or $a \cdot b$. Occasionally, we also use $\cdot$ to indicate integer multiplication, depending on context, to aid readability.
  - We do *not* use an explicit operator for matrix-matrix or matrix-vector multiplication. Examples include $\mathbf{AB}$, $\mathbf{Ab}$.
  - For matrix-scalar or vector-scalar multiplication, we explicitly write $\cdot$, e.g., $\mathbf{A} \cdot b$ and $\mathbf{a} \cdot b$.
  - We also frequently multiply coefficient vectors of polynomials with matrices, and this is denoted similarly to matrix-vector multiplication: $\mathbf{A}b$.
  - If we write $\mathbf{A} \cdot b$ where $\mathbf{A}$ is a matrix and b is a polynomial, it means the polynomial is treated as a scalar (not decomposed into its coefficient vector).
  - $\star$ denotes element-wise multiplication of vectors, such as $\mathbf{a} \star \mathbf{b}$. $\circ$ denotes dot-product between vectors or a matrix and a vector. For instance, $\mathbf{A} \circ \mathbf{b}$ is defined as $\sum_{i=0}^{m-1} \mathbf{A}_{[i]}\mathbf{b}$ where m denotes the number of rows in $\mathbf{A}$.
- Sliced access to elements of matrices, vectors, sets and polynomials:

- We write $\mathbf{A}_{[:][i]}$ to access the i-th column of a matrix $\mathbf{A}$. The same notation also applies to vectors of polynomials. For a vector of polynomials $\mathbf{a}$, $\mathbf{a}_{[:][i]}$ denotes the vector formed by the i-th coefficient of each polynomial in $\mathbf{a}$.

- For elements of matrices, vectors, sets and polynomials, we write $:m$ in the sub-index to denote the first m elements, such as $a_{[:m]}$.

- Operators:

  - $||\cdot||_\infty$ denotes the infinity norm, *i.e.*, the largest absolute value among the coefficients of a polynomial, as in $||a||_\infty$, or the largest absolute value among the elements of a vector, as in $||\mathbf{a}||_\infty$. In the case of nested structures, such as vectors of polynomials, the infinity norm is applied recursively, following a maximum-of-maximums approach.

  - $||$ denotes the concatenation operator. For operands a and b, $a||b$ represents the binary string formed by appending the binary representation of b to that of a.

  - Concatenation of matrices or vectors is denoted by $|$. For example, $\mathbf{A}|\mathbf{B}$ represents the matrix formed by concatenating the columns of $\mathbf{A}$ and $\mathbf{B}$, and $\mathbf{a}|\mathbf{b}$ denotes the vector formed by concatenating $\mathbf{a}$ and $\mathbf{b}$. In some cases, the operator may be omitted for matrices when the context clearly indicates both horizontal and vertical concatenation.

- Modular arithmetic:

  - ${}^{\pm}\mathbb{Z}_q$ denotes the set of centrally represented integers modulo q, that is, the interval $[-q/2, q/2]$ for odd q and $[-q/2, q/2+1]$ for even q. The notation $\text{mod}^{\pm} q$ refers to central (*a.k.a.* centered) modular reduction onto this range. In contrast, $\mathbb{Z}_q$ denotes the set of integers modulo q with the standard (unsigned) representatives in $[0, q)$. We use $\text{mod}^{+} q$ to indicate reduction onto $[0, q)$, and simply write $\text{mod} q$ when the reduction range is unimportant.

3.2. Lattice-Based Cryptography

Crystals-Kyber (Bos et al., 2018) and Crystals-Dilithium (Ducas et al., 2018) are examples of Lattice-Based Cryptography (LBC). They are among the first Post-Quantum Cryptography (PQC) standards selected by NIST, now standardized under the names ML-KEM and ML-DSA, respectively. The main mathematical object is the ring of polynomials $\mathcal{R}_{q,n} = \mathbb{Z}_{q,n}/(x^n + 1)$, where n is referred to as the ring modulus which is a power-of-two, and q is referred to as the coefficient modulus which is a prime number. In $\mathcal{R}_{q,n}$, each element is a polynomial of degree $n - 1$ whose coefficients are in $\mathbb{Z}_q$. The security of Kyber and Dilithium relies on the Module Learning With Errors (M-LWE) problem introduced in Langlois & Stehlé (2015), which is a module-based generalization of the Ring Learning With Errors (R-LWE) problem (Regev, 2009).

3.2.1. Crystals-Kyber

Crystals-Kyber (Bos et al., 2018) is a Post-Quantum Key Encapsulation Mechanism (KEM). A KEM is a public-key cryptographic primitive designed to securely transmit a symmetric key—hence the term key encapsulation. The encapsulated (i.e., encrypted) key is typically used for symmetric encryption of data. Encapsulation is performed using the recipient's public key, while decapsulation (i.e., decryption) is performed using the corresponding private key. Kyber is based on the M-LWE version of the well-known LPR scheme presented by Lyubashevsky, Peikert & Regev (2010). Kyber provides two layers of functions for key generation, encryption, and decryption: IND-CPA and IND-CCA functions. The IND-CCA functions internally use the IND-CPA functions and apply the Fujisaki–Okamoto (FO) transform (Fujisaki & Okamoto, 1999) to achieve IND-CCA security.

Table 3.1 Kyber parameter sets

	NIST Security Level	n	k	q	η_1	η_2	(d_u, d_v)
Kyber512	1	256	2	3329	3	2	$(10, 4)$
Kyber768	3	256	3	3329	2	2	$(10, 4)$
Kyber1024	5	256	4	3329	2	2	$(10, 5)$

Algorithm 1 $\mathsf{Kyber.CPAPKE.KeyGen}()$

Output: Public Key pk
Output: Secret Key sk

$d \leftarrow \{0,1\}^{256}$
$(\rho, \sigma) = \mathsf{G}(d)$
$\hat{\mathbf{A}} \in \mathcal{R}_{q,n}^{k \times k} = \mathsf{Parse}(\mathsf{XOF}(\rho))$ ▷ $\mathbf{A}$ is generated in NTT domain, $\hat{\mathbf{A}} = \mathsf{NTT}(\mathbf{A})$
$\mathbf{s} \in \mathcal{R}_{q,n}^{k} = \mathsf{CBD}_{\eta_1}(\mathsf{XOF}(\sigma||0))$
$\mathbf{e} \in \mathcal{R}_{q,n}^{k} = \mathsf{CBD}_{\eta_1}(\mathsf{XOF}(\sigma||k))$ ▷ Coefficients of $\mathbf{s}$ and $\mathbf{e}$ are in $[-\eta_1, \eta_1]$
$\mathbf{t} = \mathbf{A}\mathbf{s} + \hat{\mathbf{e}}$
$pk = (\hat{\mathbf{t}}, \rho)$ ▷ $\mathbf{t}$ is stored in NTT domain, $\hat{\mathbf{t}} = \mathsf{NTT}(\mathbf{t})$
$sk = \hat{\mathbf{s}}$ ▷ $\mathbf{s}$ is stored in NTT domain, $\hat{\mathbf{s}} = \mathsf{NTT}(\mathbf{s})$
return (pk, sk)

Algorithm 1 presents the IND-CPA key generation, function of Kyber. The secret and public key pair is generated using the M-LWE equation $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$ where $(\mathbf{t}, \mathbf{A})$ is public and $\mathbf{s}$ is the vector of secret polynomials. The vector of polynomials $\mathbf{e}$ represents the error term and is discarded after the key generation process. Coefficients of $\mathbf{s}$ and $\mathbf{e}$ are short. In Kyber, such short coefficients are pseudo-randomly sampled using a centered binomial distribution (CBD). Specifically, the output of CBD_{η_1} lies in the range $[-\eta_1, \eta_1]$, with values closer to 0 being more likely to occur. Note that the polynomial multiplications are performed using the NTT algorithm, which is explained in Section 3.3. Indeed, $\mathbf{t}$ and $\mathbf{s}$ are returned in their NTT representations. Table 3.1 demonstrates the parameter sets used by Kyber based on different NIST security levels.

Algorithm 2 Kyber.CPAPKE.Enc(pk, m, r)

Input: Public Key $pk = (\hat{\mathbf{t}}, \rho)$
Input: Plaintext $m \in \{0,1\}^{256}$
Input: Random Coins r
Output: Ciphertext c

1: $\hat{\mathbf{A}} \in \mathcal{R}_{q,n}^{k \times k} = \mathsf{Parse}(\mathsf{XOF}(\rho))$
2: $\mathbf{r} \in \mathcal{R}_{q,n}^{k} = \mathsf{CBD}_{\eta_1}(\mathsf{XOF}(r||0))$ ▷ Coefficients of $\mathbf{r}$ are in $[-\eta_1, \eta_1]$
3: $\mathbf{e_1} \in \mathcal{R}_{q,n}^{k} = \mathsf{CBD}_{\eta_2}(\mathsf{XOF}(r||k))$
4: $e_2 \in \mathcal{R}_{q,n} = \mathsf{CBD}_{\eta_2}(\mathsf{PRF}(r||2k))$ ▷ Coefficients of $\mathbf{e_1}$ and $\mathbf{e_2}$ are in $[-\eta_2, \eta_2]$
5: $\mathbf{u} = \mathbf{A}^T\mathbf{r} + \mathbf{e_1}$
6: $v = \mathbf{t}^T\mathbf{r} + e_2 + \mathsf{Decompress}_q(\mathsf{Decode}_1(m), 1)$
7: $\mathbf{c}_1 = \mathsf{Compress}_q(\mathbf{u}, d_u)$
8: $c_2 = \mathsf{Compress}_q(v, d_v)$
9: **return** $c = (\mathbf{c}_1, c_2)$

Algorithm 3 Kyber.CPAPKE.Dec(sk, c)

Input: Secret Key $sk = (\hat{\mathbf{s}})$
Input: Ciphertext $c = (\mathbf{c}_1, c_2)$
Output: Plaintext $m \in \{0,1\}^{256}$

1: $\mathbf{u} = \mathsf{Decompress}_q(\mathbf{c}_1, d_u)$
2: $v = \mathsf{Decompress}_q(c_2, d_v)$
3: $m = \mathsf{Encode}_1(\mathsf{Compress}_q(v - \mathbf{s}^T\mathbf{u}), 1))$
4: **return** m

Algorithm 2 and Algorithm 3 present the IND-CPA encryption and decryption functions of Kyber. During encryption, an ephemeral secret vector of polynomials $\mathbf{r} \in \mathcal{R}_{q,n}^{k}$ is generated whose coefficients are also pseudo-randomly sampled using CBD_{η_1}. Observe that the equations in lines 5-6 form another M-LWE instances ensuring the security of the output ciphertext. Both components of the ciphertext are post-processed using specialized compression functions. In particular, $\mathsf{Compress}_q$, $\mathsf{Decompress}_q$ are helper functions designed to reduce the size of the ciphertext without compromising the correctness of decryption. Observe that the output ciphertext is compressed in lines 7-8 of Algorithm 2 and decompressed before processing in lines 1-2 of Algorithm 3. The compression ratio is decided by a parameter, which is d_u or d_v.

Similarly, the functions Encode and Decode are used to convert between polynomials and binary strings. In particular, $\mathsf{Decompress}_q(\mathsf{Decode}_1(m), 1)$ maps $m \in \{0,1\}^{256}$ to $m' \in \mathcal{R}_{q,n}$ as follows:

$$m'_{[i]} = \begin{cases} \left\lfloor \frac{q}{2} \right\rfloor, & \text{if } m_{[i]} = 1 \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

The inverse transformation $\mathsf{Encode}_1(\mathsf{Compress}_q(m'),1))$ effectively rounds the input to suppress noise introduced during encryption.

$$m_{[i]} = \begin{cases} 1, & \text{if } \left\lfloor \frac{q}{4} \right\rfloor \leq m'_{[i]} < \left\lfloor \frac{3q}{4} \right\rfloor \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

The correctness of the decryption is easy to notice. Omitting the compression and decompression functions:

$$m'' = v - \mathbf{s}^T\mathbf{u} \tag{3.3}$$

$$m'' = \mathbf{t}^T\mathbf{r} + e_2 + m' - \mathbf{s}^T(\mathbf{A}^T\mathbf{r} + \mathbf{e_1}) \tag{3.4}$$

$$m'' = \mathbf{t}^T\mathbf{r} + e_2 + m' - (\mathbf{s}^T\mathbf{A}^T)\mathbf{r} + \mathbf{s}^T\mathbf{e_1} \tag{3.5}$$

$$m'' = \mathbf{t}^T\mathbf{r} + e_2 + m' - (\mathbf{t}^T - \mathbf{e}^T)\mathbf{r} + \mathbf{s}^T\mathbf{e_1} \tag{3.6}$$

$$m'' = \mathbf{t}^T\mathbf{r} + e_2 + m' - \mathbf{t}^T\mathbf{r} + \mathbf{e}^T\mathbf{r} + \mathbf{s}^T\mathbf{e_1} \tag{3.7}$$

$$m'' = e_2 + m' + \mathbf{e}^T\mathbf{r} + \mathbf{s}^T\mathbf{e_1} \tag{3.8}$$

Round as explained above suppresses the terms with small coefficients (e_2, $\mathbf{e}^T\mathbf{r}$, $\mathbf{s}^T\mathbf{e_1}$) leading to $m'' = m'$.

Algorithm 4 Kyber.CCAKEM.KeyGen()

Output: Public Key pk
Output: Secret Key sk
1: $z \leftarrow \{0,1\}^{256}$
2: $(pk, sk') = \mathsf{Kyber.CPAPKE.KeyGen}()$
3: $sk = (sk'||pk||\mathsf{H}(pk)||z)$
4: **return** (pk, sk)

Algorithm 5 Kyber.CCAKEM.Enc(pk)

Input: Public Key pk
Output: Ciphertext c
Output: Shared Key k
1: $m \leftarrow \{0,1\}^{256}$
2: $m = \mathsf{H}(m)$
3: $(k', r) = \mathsf{G}(m||\mathsf{H}(pk))$
4: $c = \mathsf{Kyber.CPAPKE.Enc}(pk, m, r)$
5: $k = \mathsf{KDF}(k'||\mathsf{H}(c))$
6: **return** (c, k)

Algorithm 6 $\mathsf{Kyber.CCAKEM.Dec}(c, sk, pk)$

Input: Ciphertext c
Input: Secret Key sk
Output: Shared Key k

1: $(sk'||pk||\mathsf{H}(pk)||z) = sk$
2: $m' = \mathsf{Kyber.CPAPKE.Dec}(sk', c)$
3: $(k', r) = \mathsf{G}(m'||h)$
4: $c' = \mathsf{Kyber.CPAPKE.Enc}(pk, m', r)$
5: **if** $c = c'$ **then**
6: **return** $k = \mathsf{KDF}(k'||\mathsf{H}(c))$
7: **else**
8: **return** $k = \mathsf{KDF}(k||\mathsf{H}(c))$
9: **end if**

Algorithm 4, Algorithm 5 and Algorithm 6 present the IND-CCA key generation, encryption, and decryption functions of Kyber. Observe from Algorithm 6 that the FO transform re-encrypts the initially decrypted m' and compares the result of re-encryption c' with the input ciphertext c. In this way, chosen ciphertext attacks are mitigated by preventing the acceptance of maliciously crafted ciphertexts.

H, G, XOF, PRF, and KDF are hash-based functions which are from the NIST FIPS-202 standard (Dworkin & others, 2015). H and G are used for hashing, instantiated as $\mathsf{SHA3-256}$ and $\mathsf{SHA3-512}$, respectively. XOF is instantiated as $\mathsf{SHAKE-128}$; PRF, and KDF are instantiated as $\mathsf{SHAKE-256}$.

3.2.2. Crystals-Dilithium

Crystals-Dilithium (Ducas et al., 2018) is a Post-Quantum Digital Signature Algorithm (DSA). A DSA is a public-key cryptographic scheme in which signatures are generated using the secret key and verified using the corresponding public key.

Algorithm 7 presents the key generation function of Dilithium. As in Kyber, key generation in Dilithium is also based on the M-LWE equation $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$. In contrast, the error term is retained as a part of the secret key in Dilithium and is denoted by $\mathbf{s}_2$. The coefficients of the polynomials in $\mathbf{s}_1$ and $\mathbf{s}_2$ are short like Kyber. In this case, they are sampled uniformly at random from the interval $[-\eta, \eta]$. The uniform and pseudo-random sampling function used for this purpose is denoted by U_η, as shown in line 3 of Algorithm 7. On the other hand, $\mathbf{A}$ is uniformly distributed in $\mathcal{R}_{q,n}$, pseudo-randomly generated using $\mathsf{ExpandA}$ function. Table 3.2

Algorithm 7 Dilithium.KeyGen()

Output: Public Key pk
Output: Secret Key sk
1: $\zeta \leftarrow \{0,1\}^{256}$
2: $(\rho, \rho', k) = \mathsf{H}(\zeta)$
3: $(\mathbf{s_1}, \mathbf{s_2}) \in \mathcal{R}_{q,n}^{\ell} \times \mathcal{R}_{q,n}^{k} = \mathsf{U}_{\eta}(\mathsf{H}(\rho'))$ ▷ Coefficients of $\mathbf{s_1}$ and $\mathbf{s_1}$ are uniform in $[-\eta, \eta]$
4: $\hat{\mathbf{A}} \in \mathcal{R}_{q,n}^{k \times \ell} = \mathsf{ExpandA}(\rho)$ ▷ $\mathbf{A}$ is generated in NTT domain, $\hat{\mathbf{A}} = \mathsf{NTT}(\mathbf{A})$
5: $\mathbf{t} = \mathbf{A}\mathbf{s_1} + \mathbf{s_2}$
6: $(\mathbf{t_1}, \mathbf{t_0}) = \mathsf{Power2Round}_q(\mathbf{t}, d)$
7: $tr = \mathsf{H}(\rho || \mathbf{t_1})$
8: $pk = (\rho, \mathbf{t_1})$
9: $sk = (\rho, k, tr, \mathbf{s_1}, \mathbf{s_2}, \mathbf{t_0})$
10: **return** (pk, sk)

Table 3.2 Dilithium parameter sets. λ denotes the NIST security level.

	λ	n	(k,ℓ)	q	η	d	γ_1	γ_2	β	ω	τ
Dilithium2	2	256	(4,4)	8380417	2	13	2^{17}	$(q-1)/88$	78	80	39
Dilithium3	3	256	(6,5)	8380417	4	13	2^{19}	$(q-1)/32$	196	55	49
Dilithium5	5	256	(8,7)	8380417	2	13	2^{19}	$(q-1)/32$	120	75	60

demonstrates the parameter sets employed by Dilithium, corresponding to different security levels.

Algorithm 8 and Algorithm 9 present the signature generation and verification functions of Dilithium, respectively. The signature process employs the rejection sampling idea. This means, the main loop signature algorithm iterates (lines 6-23) until a valid signature is found which passes the correctness and security checks. The first check in line 14 ensures security, while the second check ensures both security and correctness. At each iteration, a masking vector of polynomials $\mathbf{y}$ and a challenge polynomial c are sampled. The coefficients of $\mathbf{y}$ are uniform in $(-\gamma_1, \gamma_1]$. On the other hand, c has exactly τ number of non-zero coefficients, which are either 1 or -1. A candidate signature is computed as $\mathbf{z} = \mathbf{y} + c \cdot \mathbf{s_1}$. The parameters are chosen so that the expected number of iterations remains low, specifically, 4.25, 5.1, and 3.85 for Dilithium2, Dilithium3, and Dilithium5, respectively.

$\mathsf{Power2Round}_q$ is a supporting function that is used to split $\mathbf{t}$ into its lower-order bits $\mathbf{t_0}$ and higher-order bits $\mathbf{t_1}$, satisfying $\mathbf{t} = \mathbf{t_1} \cdot 2^d + \mathbf{t_0}$. Similarly to compression in Kyber, $\mathsf{Power2Round}_q(., d)$ reduces the size of the public key based on the parameter d. This optimization is based on the observation that $\mathbf{A}\mathbf{z} - c \cdot \mathbf{t}$ does not significantly depend on the lower-order bits $\mathbf{t_0}$ of $\mathbf{t}$. Therefore, the verifier only uses $\mathbf{t_1}$, as shown in line 4 of Algorithm 9. Accordingly, the signer provides a set of "hints" $\mathbf{h}$ in the output of the signature generation. The hint $\mathbf{h}$ correspond to the carry

Algorithm 8 Dilithium.Sign(sk, m)

Input: Secret Key $sk = (\rho, k, tr, \mathbf{s_1}, \mathbf{s_2}, \mathbf{t_0})$
Input: Message m
Output: Signature σ

$\hat{\mathbf{A}} \in \mathcal{R}_{q,n}^{k \times \ell} = \mathsf{ExpandA}(\rho)$
$\mu = \mathsf{H}(tr||m)$
$\rho' = \mathsf{H}(k||\mu)$
$\kappa = 0$
$(\mathbf{z}, \mathbf{h}) = \perp$
while $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
    $\mathbf{y} \in \mathcal{R}_{q,n}^{\ell} = \mathsf{ExpandMask}(\rho', \kappa)$ ▷ Coefficients of $\mathbf{y}$ are in $(-\gamma_1, \gamma_1]$
    $\mathbf{w} = \mathbf{Ay}$
    $\mathbf{w_1} = \mathsf{HighBits}_q(\mathbf{w}, 2\gamma_2)$
    $\tilde{c} = \mathrm{H}(\mu||\mathbf{w_1})$
    $c \in \mathcal{R}_{q,n} = \mathsf{SampleInBall}(\tilde{\mathsf{c}})$ ▷ c has exactly τ #non-zero coefficients which are ± 1
    $\mathbf{z} = \mathbf{y} + c \cdot \mathbf{s_1}$
    $\mathbf{r_0} = \mathsf{LowBits}_q(\mathbf{w} - c \cdot \mathbf{s_2}, 2\gamma_2)$
    if $(||\mathbf{z}||_\infty \geq \gamma_1 - \beta)$ or $(||\mathbf{r_0}||_\infty \geq \gamma_2 - \beta)$ **then**
        $(\mathbf{z}, \mathbf{h}) = \perp$
    else
        $\mathbf{h} = \mathsf{MakeHint}_q(-c \cdot \mathbf{t_0}, \mathbf{w} - c \cdot \mathbf{s_2} + c \cdot \mathbf{t_0}, 2\gamma_2)$
        if $(||c\mathbf{t_0}||_\infty \geq \gamma_2)$ or (# of 1's in $\mathbf{h} > \omega$) **then**
            $(\mathbf{z}, \mathbf{h}) = \perp$
        end if
    end if
    $\kappa = \kappa + \ell$
end while
return $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$

Algorithm 9 Dilithium.Verify(pk, m, σ)

Input: Public Key $pk = (\rho, \mathbf{t_1})$
Input: Message m
Input: Signature $(\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h}))$
Output: Boolean result indicating the input signature is valid or not

$\hat{\mathbf{A}} \in \mathcal{R}_{q,n}^{k \times \ell} = \mathsf{ExpandA}(\rho)$
$\mu = \mathsf{H}(\mathsf{H}(\rho||\mathbf{t_1})||m)$
$c = \mathsf{SampleInBall}(\tilde{\mathsf{c}})$
$\mathbf{w_1}' = \mathsf{UseHint}_q(\mathbf{h}, \mathbf{Az} - c\mathbf{t_1} \cdot 2^d, 2\gamma_2)$
$\tilde{c}' = \mathsf{H}(\mu||\mathbf{w_1}')$
if $(||\mathbf{z}||_\infty < \gamma_1 - \beta)$ and (# of 1's in $\mathbf{h} < \omega$) and $(\tilde{c} = \tilde{c}')$ **then**
    return True
else
    return False
end if

positions caused by the addition of the omitted term $c\mathbf{t_0}$. $\mathsf{MakeHint}_q$ and $\mathsf{UseHint}_q$ complement each other for generating and using the hints. The checks in line 18 of Algorithm 8 are needed for the correctness of this hint mechanism. For $a, b \in \mathcal{R}_{q,n}$ with $||a|| \leq \alpha/2$, $\mathsf{MakeHint}_q$, $\mathsf{UseHint}_q$ and $\mathsf{HighBits}_q$ satisfy the following:

$$\mathsf{UseHint}_q(\mathsf{MakeHint}_q(a, b, \alpha), b, \alpha) = \mathsf{HighBits}_q(a + b, \alpha) \tag{3.9}$$

In a nutshell, for $x \in \mathbb{Z}_q$ and positive integer α, the functions $\mathsf{HighBits}_q(x, \alpha)$ and $\mathsf{LowBits}_q(x, \alpha)$ are defined as:

$$\mathsf{HighBits}_q(x, \alpha) = -\lfloor -x/\alpha \rceil \tag{3.10}$$

$$\mathsf{LowBits}_q(x, \alpha) = x - (-\lfloor -x/\alpha \rceil) \cdot \alpha \pmod{q} \tag{3.11}$$

with a pre-defined special case for both.

The correctness of Dilithium's verification is slightly more difficult to understand compared to decryption of Kyber. The verifier computes:

$$\mathbf{w_1}' = \mathsf{UseHint}_q(\mathbf{h}, \mathbf{Az} - c \cdot \mathbf{t_1} \cdot 2^d, 2\gamma_2) \tag{3.12}$$

$$\mathbf{w_1}' = \mathsf{UseHint}_q(\mathbf{h}, \mathbf{A}(\mathbf{y} - c \cdot \mathbf{s_1}) - c \cdot \mathbf{t_1} \cdot 2^d - c \cdot \mathbf{t_0} + c \cdot \mathbf{t_0}, 2\gamma_2) \tag{3.13}$$

$$\mathbf{w_1}' = \mathsf{UseHint}_q(\mathbf{h}, \mathbf{w} + c \cdot \mathbf{As_1} - c \cdot \mathbf{t} + c \cdot \mathbf{t_0}, 2\gamma_2) \tag{3.14}$$

$$\mathbf{w_1}' = \mathsf{UseHint}_q(\mathbf{h}, \mathbf{w} + c \cdot (\mathbf{As_1} - \mathbf{t}) + c \cdot \mathbf{t_0}, 2\gamma_2) \tag{3.15}$$

$$\mathbf{w_1}' = \mathsf{UseHint}_q(\mathbf{h}, \mathbf{w} - c \cdot \mathbf{s_2} + c \cdot \mathbf{t_0}, 2\gamma_2) \tag{3.16}$$

Applying line 23 of Algorithm 8 and Equation (3.2.2):

$$\mathbf{w_1}' = \mathsf{UseHint}_q(\mathsf{MakeHint}_q(-c \cdot \mathbf{t_0}, \mathbf{w} - c \cdot \mathbf{s_2} + c\mathbf{t_0}, 2\gamma_2), \mathbf{w} - c\mathbf{s_2} + c \cdot \mathbf{t_0}, 2\gamma_2) \tag{3.17}$$

$$\mathbf{w_1}' = \mathsf{HighBits}_q(\mathbf{w} - c \cdot \mathbf{s_2}, 2\gamma_2) \tag{3.18}$$

β satisfies $||c \cdot \mathbf{s_2}||_\infty \leq \beta$ by definition and the signer checks $||\mathsf{LowBits}_q(\mathbf{w} - c \cdot \mathbf{s_2}, 2\gamma_2)||_\infty < \gamma_2 - \beta$ in line 14 of Algorithm 8. As a result:

$$||\mathsf{LowBits}_q(\mathbf{w} - c \cdot \mathbf{s_2}, 2\gamma_2) + c \cdot \mathsf{s}_2||_\infty < \gamma_2 \tag{3.19}$$

Verbally, adding $c \cdot \mathsf{s}_2$ to $\mathbf{w} - c \cdot \mathbf{s_2}$ does not change its higher-order bits, considering

decomposition with γ_2:

$$\mathsf{HighBits}_q(\mathbf{w} - c \cdot \mathbf{s_2}, 2\gamma_2) = \mathsf{HighBits}_q(\mathbf{w} - c \cdot \mathbf{s_2} + c \cdot \mathbf{s_2}, 2\gamma_2) \tag{3.20}$$

$$= \mathsf{HighBits}_q(\mathbf{w}, 2\gamma_2) \tag{3.21}$$

$$= \mathbf{w_1} \tag{3.22}$$

$$\mathbf{w_1}' = \mathbf{w_1} \tag{3.23}$$

3.3. Number Theoretic Transform

Number Theoretic Transform (NTT) is a special form of Fast-Fourier Transform (FFT) that operates over $\mathbb{Z}_q$ instead of complex numbers $\mathbb{C}$. NTT allows efficient multiplication of polynomials over $\mathcal{R}_{q,n}$. Representing a polynomial $a(x) \in \mathcal{R}_{q,n}$ in the NTT domain can be viewed as an application of CRT. Polynomial multiplication is achieved by element-wise multiplying the NTT representations of the operands:

$$a(x) \cdot b(x) = \mathsf{iNTT}(\mathsf{NTT}(a(x)) \star \mathsf{NTT}(b(x))) \tag{3.24}$$

In order NTT to be defined in $\mathcal{R}_{q,n}$, the following condition must be satisfied:

$$q \equiv 1 \pmod{2n} \tag{3.25}$$

This ensures a primitive $2n$-th root of unity ψ_{2n} exists in $\mathbb{Z}_q$ for which $\psi_{2n}^n = -1 \bmod q$. This variant of NTT is also referred to as the *negacyclic NTT*. Dilithium's parameter selection (see Table 3.2) allows the use of a negacyclic NTT as $8380417 \equiv 1 \pmod{256}$.

Using ψ_{2n}, $(x^n + 1)$ is factored in to $\prod_{i=0}^{n-1}(x - \psi_{2n}^{2i+1})$. NTT computes the following isomorphism:

$$a(x) \approx \mathsf{NTT}(a(x)) : \mathbb{Z}_q[x]/(x^n + 1) \rightarrow \prod_{i=0}^{n-1} \mathbb{Z}_q[x]/(x - \psi_{2n}^{2i+1})$$

Here, the i-th element of $\mathsf{NTT}(a(x))$ is the remainder from dividing $a(x)$ to $(x - \psi_{2n}^{2i+1})$.

NTT can be computed by evaluating the polynomial at the powers of ψ_{2n}. Let

$\hat{a} = \mathsf{NTT}(a(x))$ for $a(x) \in \mathcal{R}_{q,n}$ and $\hat{a} \in \mathbb{Z}_q$. Then, forward and backward NTTs are defined as follows:

$$\hat{a}_{[i]} = \sum_{j=0}^{n-1} a_{[j]} \cdot \psi_{2n}^{j(2i+1)} = a(\psi_{2n}^{2i+1}) \qquad \text{for } 0 \leq i < n \tag{3.26}$$

$$a_{[i]} = \sum_{j=0}^{n-1} \hat{a}_{[j]} \cdot \psi_{2n}^{-(j(2i+1))} \qquad \text{for } 0 \leq i < n \tag{3.27}$$

Notice that both the forward and inverse transformations can be viewed as matrix-vector multiplication:

$$\hat{a} = \mathbf{\Omega}_{q,n} a \tag{3.28}$$

$$a = \mathbf{\Omega}_{q,n}^{-1} \hat{a} \tag{3.29}$$

where $\mathbf{\Omega}_{q,n} \in \mathbb{Z}_q^{n \times n}$ and $\mathbf{\Omega}_{q,n}^{-1} \in \mathbb{Z}_q^{n \times n}$ referred to as forward and backward NTT matrices for $\mathcal{R}_{q,n}$, respectively. $\mathbf{\Omega}_{q,n[i][j]} = \psi_{2n}^{j(2i+1)}$ and $\mathbf{\Omega}_{q,n[i][j]}^{-1} = \psi_{2n}^{-j(2i+1)}$ for $0 \leq i, j < n$.

These calculations require $\Theta(n^2)$ steps. In case n is a power of 2, NTT can be computed efficiently by splitting the polynomial to half of its size in a recursive manner until a linear degree is reached. The transformation in each layer can be efficiently implemented using *Cooley-Tukey* (CT) butterfly circuit (Cooley & Tukey, 1965). For degree-$n/2^i$ polynomial $a(x) = a_0(x) + a_1(x) \cdot x^{n/2^{i+1}}$, the CT butterfly is defined by the map

$$a_0(x) + a_1(x) \cdot x^{n/2^{i+1}} \rightarrow \Big[a_0(x) + \delta \cdot a_1(x), a_0(x) - \delta \cdot a_1(x)\Big] \tag{3.30}$$

where δ is a power of ψ_{2n}, called the *twiddle factor*. In this manner, full NTT computation requires $\log n$ layers. For the inverse transformation, most applications use the *Gentleman-Sande* (GS) butterfly (Gentleman & Sande, 1966):

$$\Big[a_0(x), a_1(x)\Big] \rightarrow \Big((a_0(x) + a_1(x)) \cdot 2^{-1}\Big) + \Big((a_0(x) - a_1(x)) \cdot 2^{-1} \cdot \delta^{-1}\Big) \cdot x^{n/2^{i+1}} \tag{3.31}$$

Using CT or GS when n is a power of 2, computing forward or backward NTT takes $\Theta(n \log n)$ steps.

As previously stated, multiplication in the NTT domain is performed element-wise. This operation is often referred to as the *base multiplication*. For $\hat{a} = \mathsf{NTT}(a)$,

$\hat{b} = \mathsf{NTT}(a)$, and $\hat{c} = \hat{a} \star \hat{b}$:

$$c_{[i]} = a_{[i]} \cdot b_{[i]} \pmod{q} \qquad \text{for } 0 \leq i < n \tag{3.32}$$

3.3.1. Incomplete NTT

For efficiency reasons, or due to constraints imposed by the parameters q and n, the NTT can be computed using only $m < \log n$ layers. In this case, polynomials are recursively split into degree-$n/2^m$ polynomial factors, denoted by $\mathsf{NTT}_m(a(x))$. A necessary condition for applying NTT_m in the negacyclic setting is that

$$q \equiv 1 \pmod{\frac{n}{2^{\log n - m - 1}}} \tag{3.33}$$

as discussed by (Lyubashevsky & Seiler, 2019b).

For instance, Kyber's parameter selection (see Table 3.1) allows an incomplete NTT of $m = \log n - 1 = 7$. Because $3329 \not\equiv 1 \pmod{256}$ but $3329 \equiv 1 \pmod{128}$.

When $m = \log n - 1$ as in Kyber, NTT representations of polynomials are vectors of degree-1 polynomials with $n/2$ elements. Specifically, for $a(x) \in \mathcal{R}_{q,n}$:

$$\hat{a} = \mathsf{NTT}_{\log n - 1}(a(x)) \in \prod_{i=0}^{n/2-1} \frac{\mathbb{Z}_q}{(x^2 - \psi_n^{2i+1})} \tag{3.34}$$

As a result, the base multiplication refers to multiplication of degree-1 polynomials. Let $\hat{a} = \mathsf{NTT}_m(a)$, $\hat{b} = \mathsf{NTT}_m(b)$ and $\hat{c} = \hat{a} \star \hat{b}$. For $0 \leq i < n/2$:

$$\hat{c}_{[i][0]} = \hat{a}_{[i][0]} \cdot \hat{b}_{[i][0]} + \hat{a}_{[i][0]} \cdot \hat{b}_{[i][0]} \cdot \zeta_i \pmod{q} \tag{3.35}$$

$$\hat{c}_{[i][1]} = \hat{a}_{[i][0]} \cdot \hat{b}_{[i][1]} + \hat{a}_{[i][1]} \cdot \hat{b}_{[i][0]} \pmod{q} \tag{3.36}$$

where $\zeta_i = \psi_n^{2i+1}$.

In this setting, forward and backward NTTs are applied independently to even and odd degree coefficients of a. Let $a_0 = \sum_{i=0}^{n/2-1} a_{[2i]} x^{2i}$ and $a_1 = \sum_{i=0}^{n/2-1} a_{[2i+1]} x^{2i+1}$.

Table 3.3 Carrier primes used in incomplete NTT based implementation of Dilithium.

	n	q'
Dilithium2	256	257
Dilithium3	256	769
Dilithium5	256	257

Using Equation (3.29),

$$\hat{a}_{[:][0]} = \mathbf{\Omega}_{q,m} a_0 \qquad \hat{a}_{[:][1]} = \mathbf{\Omega}_{q,m} a_1 \tag{3.37}$$

$$a_0 = \mathbf{\Omega}_{q,m}^{-1} \hat{a}_{[:][0]} \qquad a_1 = \mathbf{\Omega}_{q,m}^{-1} \hat{a}_{[:][1]} \tag{3.38}$$

To unify the notation with Equation (3.29), these transformations can be merged into a single operation by duplicating the elements of $\mathbf{\Omega}_{q,m}$ and $\mathbf{\Omega}_{q,m}^{-1}$ to form matrices in $\mathbb{Z}_q^{n \times n}$. When the NTT is incomplete, and we refer to $\mathbf{\Omega}_{q,n}$ or $\mathbf{\Omega}_{q,n}^{-1}$, we mean the matrix resulting from this duplication.

3.3.2. Dilithium's Incomplete NTT based implementation

As previously discussed, incomplete NTT is sometimes preferred for efficiency reasons, even when the parameter set would allow the use of a complete NTT. Abdulrahman et al. (2022) presented an incomplete NTT based implementation of Dilithium. In their approach, incomplete NTT arithmetic is used for certain operations, while others still rely on the complete NTT. In particular, the authors show that the multiplications $c \cdot \mathbf{s_1}$ and $c \cdot \mathbf{s_2}$ can be carried out using a carrier prime and incomplete NTT arithmetic. Recall that c is a challenge polynomial with exactly τ non-zero coefficients, each equal to ± 1. Meanwhile, the coefficients of polynomials in $\mathbf{s_1}$ and $\mathbf{s_2}$ are bounded in absolute value by η. Consequently, the carrier prime, denoted by q', must satisfy:

$$q' > 2\tau\eta \tag{3.39}$$

The choice of carrier prime for each security level of Dilithium is shown in Table 3.3.

Listing 3.1 Behavior of ARM `SMUAD` instruction

```
/*
 * SMUAD Rd, Rn, Rm
 * Signed Multiply Dual with Add
 */
Rd = (((int16_t)(Rn & 0xFFFF)) * ((int16_t)(Rm & 0xFFFF)))
   + (((int16_t)(Rn >>     16)) * ((int16_t)(Rm >>     16)));
```

Listing 3.2 Behavior of ARM `SMUADX` instruction

```
/*
 * SMUADX Rd, Rn, Rm
 * Signed Multiply Dual with Add
 */
Rd = (((int16_t)(Rn & 0xFFFF)) * ((int16_t)(Rm >>     16)))
   + (((int16_t)(Rn >>     16)) * ((int16_t)(Rm & 0xFFFF)));
```

Abdulrahman et al. (2022) demonstrates that the incomplete NTT arithmetic offers significant performance improvements on the Cortex-M4. For example, in the case of Dilithium3, the number of clock cycles required for the forward NTT is reduced from 8093 to 5200, resulting in a 1.56× speedup. We refer the reader to Abdulrahman et al. (2022) for further benchmarking details. A key reason for this improvement is that the incomplete NTT implementation leverages 16-bit arithmetic, as both $q' = 257$ and $q' = 769$ are 16-bit primes. In contrast, the complete NTT implementation for Dilithium relies on 32-bit arithmetic. Additionally, special ARM instructions are employed for base multiplication. Specifically, the degree-1 polynomial multiplications in Equation (3.35) and Equation (3.36) are efficiently implemented using the `SMUAD(X)`[1] instruction. The behavior of these instructions is presented in Listing 3.1 and Listing 3.2. The `SMUAD` instruction computes the product of the lower 16 bits of two 32-bit integers and adds it to the product of the upper 16 bits of the same integers. The `SMUADX` instruction computes the product of the lower 16 bits of one integer with the upper 16 bits of another integer and adds it to the product of the upper 16 bits of the first integer with the lower 16 bits of the second integer. These instructions are used to compute Equation (3.35) and Equation (3.36) efficiently.

[1] https://developer.arm.com/documentation/ddi0403/d/Application-Level-Architecture/Instruction-Details/Alphabetical-list-of-ARMv7-M-Thumb-instructions/SMUAD–SMUADX

As follows, we briefly review the modular reduction techniques that are adapted in LBC, which serve as the core building blocks of arithmetic in $\mathbb{Z}_q$. Compared to ECC or RSA, the modular arithmetic in LBC deals with relatively shorter integers. Additionally, operating with signed integers in modular arithmetic proves to be more efficient in LBC (Abdulrahman et al., 2022; Alkim et al., 2020; Botros et al., 2019; Greconici et al., 2021; Huang et al., 2022). The main reason for this is due to the fact that it simply eliminates the need for an extra addition for preventing negativeness in the butterfly units (see Equation (3.3)). Performing signed arithmetic requires the use of central modular reductions. For an odd modulus q, a central reduction is explicitly denoted as

$$a' = a \bmod^{\pm} q \tag{3.40}$$

which maps a to the unique representative a' in the interval $[-q/2, q/2]$. In contrast, the classical (unsigned) modular reduction maps elements to the range $[0, q)$.

We should also note that, beyond implementation considerations, both Kyber and Dilithium employ signed modular arithmetic at the algorithmic level. For instance, the infinity norms frequently used in Dilithium operate on the signed modulo-q representatives of polynomial coefficients. Table 3.4 summarizes state-of-the-art reduction schemes implemented on the ARM Cortex-M4.

3.4.1. Barrett Reduction

The Barrett reduction was originally proposed by Barrett (1986). Its main idea is to subtract a factor of the modulus q from the number to reduce by approximating the division of the number by q through a pre-computed factor q' and shifting. To perform $a' = a \bmod q$ using Barrett reduction:

$$t = a \cdot q' \gg 2\beta \tag{3.41}$$

$$a' = a - t \cdot q \tag{3.42}$$

where q' is precomputed as $q' = \left\lfloor \frac{2^{2\beta}}{q} \right\rfloor$, and β is the machine word-size that satisfies $q < 2^{\beta}$.

A signed version of Barrett reduction adapted for LBC is proposed by Seiler (2018). The input range of the signed Barrett reduction is $[-\beta/2, \beta/2)$, and the output range is $[0, q]$. A 9-cycle implementation of the signed Barrett reduction for packed integers dedicated to ARM Cortex-M4 is presented by Alkim et al. (2020). Packing of integers refers to storing two $\beta = 16$-bit integers in a $2\beta = 32$-bit register, which is then passed to the packed reduction function. Later, a 6-cycle implementation of Barrett reduction for packed integers was reported by Abdulrahman et al. (2022), which performs a central reduction with an output range $[-q/2, q/2]$.

3.4.2. Montgomery Reduction

The Montgomery reduction is first proposed by Montgomery (1985). Similar to the Barrett reduction, it enables a constant time reduction by eliminating the need for division. To perform $a' = a \pmod{q}$ using the Naive Montgomery reduction, the following steps are performed:

$$t = a \cdot q' \mod 2^{\beta} \tag{3.43}$$

$$a' = (a + t \cdot q) \gg \beta \tag{3.44}$$

where q' is a precomputed constant computed as $q' = -q^{-1} \mod 2^{\beta}$, and β is the machine word size that satisfies $q < 2^{\beta}$. Note that the Montgomery reduction actually returns $a' \cdot 2^{-\beta}$, where $2^{-\beta}$ is referred to as the Montgomery factor.

A signed version of Montgomery reduction is presented by Seiler (2018), with an input range $[-q\beta/2, q\beta/2)$ and output range $(-q, q)$. While a 3-cycle implementation on ARM Cortex-M4 was initially given by Botros et al. (2019) for $\beta = 16$, the state-of-the-art implementation of Montgomery reduction (Alkim et al., 2020; Greconici et al., 2021) takes 2 cycles for both $\beta = 16$ and $\beta = 32$. 2-cycle implementation of Alkim et al. (2020) is provided in Listing 3.3. Also, an 8-cycle implementation of Montgomery reduction for packed integers was presented by Alkim et al. (2020). We would like to note that a final correction may be required after the signed Montgomery reduction to find the residue in the signed range $[-q/2, q/2]$ which is the ultimate goal.

Listing 3.3 2-cycle signed Montgomery reduction on the Cortex-M4 (Alkim et al., 2020). $\beta = 16$. Input: $-8 \cdot q \leq a < 8 \cdot q$. Output: $-q < a' < q$ where $a' \cdot 2^{-16} \equiv \mod q$.

```
1 SMULBB t, a, q'          ; t = a.q'
2 SMLABB a', t, q, a       ; a = a + t.q
```

3.4.3. Plantard Reduction

The Plantard reduction (Plantard, 2021) is a more recent algorithm compared to its counterparts, Montgomery and Barrett. While the original Plantard reduction operates on unsigned integers, Huang et al. (2022) proposed an improved version, which operates on signed integers to be employed in LBC. The output range of the signed version is $[-q/2, q/2]$, the same as the state-of-art Barrett reduction. One advantage of the Plantard reduction is that it enables 2-cycle modular multiplication by a constant, outperforming the 3-cycle Montgomery multiplication. The multiplication by a constant is beneficial for implementing the butterfly units during the NTT transformations (see Equation (3.3)). On the other hand, the improved Plantard reduction also takes 2 cycles on ARM Cortex-M4, the same as Montgomery. However, Plantard's 2-cycle implementation enables a larger input range and a smaller output range that is desirable. Specifically, the Plantard reduction generates the output in the exact range $[-q/2, q/2]$. In other words, it does not require any final correction. This is a significant improvement over the 2-cycle Montgomery reduction whose output range is $(-q, q)$. As a side note, packed reduction takes 5 cycles.

Table 3.4 Summary of state-of-the-art reduction implementations on ARM Cortex-M4.

Scheme	β	Input Range	Output Range	Packed	Cycles
Montgomery (Alkim et al., 2020)	16	$[-q\beta/2, q\beta/2)$	$(-q, q)$	✗	2
Montgomery (Greconici et al., 2021)	32	$[-q\beta/2, q\beta/2)$	$(-q, q)$	✗	2
Montgomery (Alkim et al., 2020)	16	$[-q\beta/2, q\beta/2)$	$(-q, q)$	✓	8
Barrett (Seiler, 2018)✴	16	$[-\beta/2, \beta/2)$	$[0, q]$	✗	3
Barrett (Abdulrahman et al., 2022)	16	$[-\beta/2, \beta/2)$	$[-q/2, q/2]$	✓	6
Plantard (Huang et al., 2022)	16	$[-q^2 2^{2\alpha'}, q^2 2^{2\alpha'}]$☆	$[-q/2, q/2]$	✗	2
Plantard (Huang et al., 2022)	16	$[-q^2 2^{2\alpha'}, q^2 2^{2\alpha'}]$☆	$[-q/2, q/2]$	✓	5

✴ gives the definition of the algorithm but does not present an implementation on ARM Cortex-M4.
☆ α' is a parameter of Plantard reduction that satisfies $q < 2^{\beta-\alpha'-1}$. β denotes the machine word size.

3.5. Side-Channel Analysis

In this section, we briefly discuss SCA attacks, focusing on the non-profiled class, along with their countermeasures and attacks targeting those countermeasures.

3.5.1. Overview

In general, an attacker records side-channel leakage from a victim device during the execution of cryptographic operations involving secret keys, such as symmetric key encryption/decryption, keyed hashing, signature generation (*e.g.*, Dilithium, see Algorithm 8), or secret key decryption (*e.g.*, Kyber, see Algorithm 6, Algorithm 3). Typically, the attacker also has access to metadata associated with the cryptographic process—this may be inputs or outputs such as ciphertext, plaintext,

or messages to be signed. Leveraging the side-channel data alongside this metadata, the attacker applies statistical analysis to infer the secret key. An overview of this attack methodology is illustrated in Figure 3.1. Common side-channels include timing information (Kocher, 1996), power consumption (Kocher et al., 1999), and electromagnetic (EM) emanations (Agrawal et al., 2002). This study focuses on power leakage, though the techniques discussed are also applicable to EM side-channel attacks.

One of the most important characteristics of SCA attacks is that they allow partial recovery of the secret key. These portions, referred to as *subkeys*, are small enough to make brute-force guessing feasible. Each subkey is attacked individually, and the process is typically repeated until the entire key is recovered. For example, in a common scenario, a 128-bit secret key is targeted in 8-bit subkeys, resulting in 16 independent sub-attacks.

3.5.2. Leakage Model

Power leakage-based SCA attacks exploit the fact that a processor's power consumption depends on the data it processes. We model the side-channel leakage L as a random variable composed of two parts: a data-dependent leakage function $\mathsf{L}(X)$, where $X \in \mathtt{X}$ is a random intermediate variable, and independent noise $\mathsf{N}(\mu, \sigma)$ following a Gaussian distribution. Formally, this can be expressed as:

$$L = \mathsf{L}(X) + \mathsf{N}(\mu, \sigma) \tag{3.45}$$

The most commonly observed and utilized leakage functions are the Hamming Weight and Hamming Distance functions.

3.5.3. Non-Profiled SCA Attack

In the non-profiled SCA attack setting, the attacker has access to ν side-channel leakages $\mathtt{L} = \{L^{(i)}\}_{i=0}^{\nu-1}$ and the corresponding publicly known and varying inputs/outputs to the attacked algorithm $\mathtt{C} = \{C^{(i)}\}_{i=0}^{\nu-1}$. Each index i corresponds to a

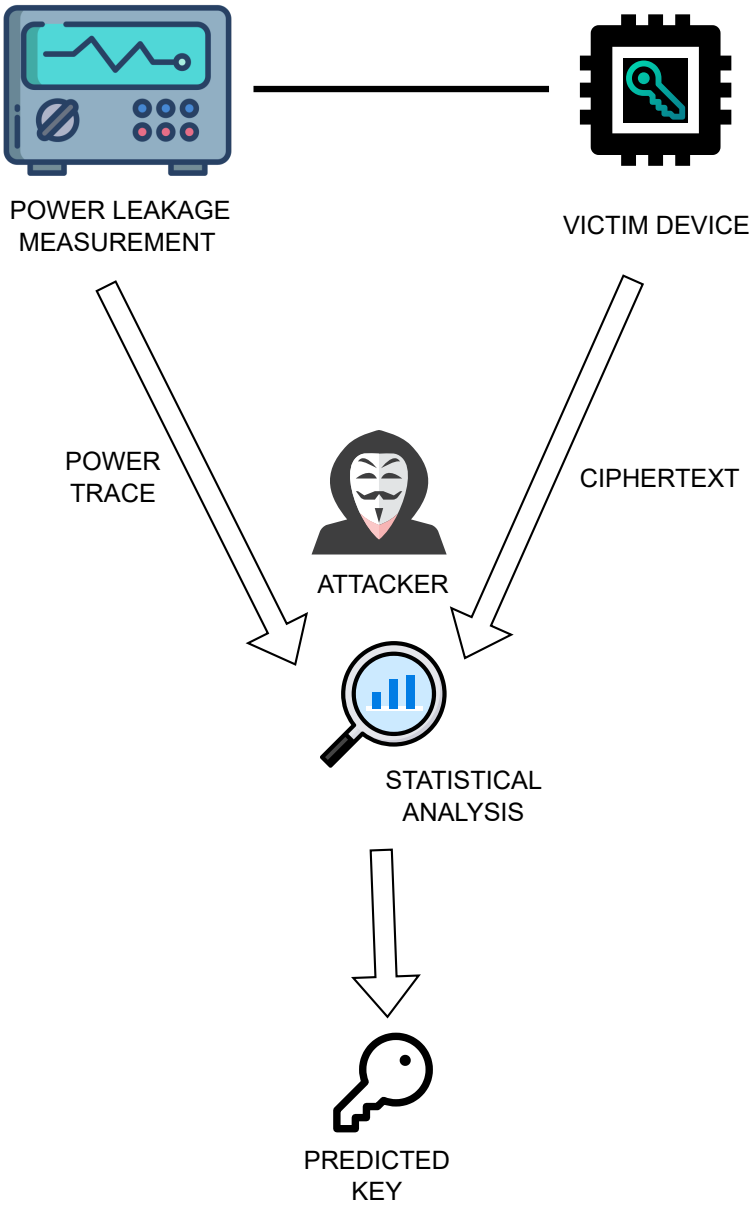


Figure 3.1 General model for power leakage based SCA attacks.

different execution of the cryptographic algorithm under attack. The known data, consisting of $\mathtt{L}$ and $\mathtt{C}$, are collectively referred to as *traces*. The steps of the non-profiled SCA attacks is summarized as follows:

- The attacker chooses a target function G that involves both the unknown secret s and known data C, with $X = \mathsf{G}(s, C)$.
- The attacker prepares a set of hypotheses (*a.k.a.* predictions) for the attacked secret, $\mathtt{K}$. For each $\mathtt{K}_{[t]}$, the attacker computes the set of hypothetical intermediates $\mathtt{X}_t = \{X_t^{(i)}\}_{i=0}^{\nu-1}$ where $X_t^{(i)} = \mathsf{G}(\mathtt{K}_{[t]}, C^{(i)})$.
- To test each hypothesis $\mathtt{K}_{[t]}$, the attacker statistically compares L with X_t over the observations $\mathtt{L}$ and $\mathtt{X}_t$, resulting in a score: $\mathsf{Score}(\mathtt{K}_{[t]}) = \mathsf{D}(L, X_t)$. The statistical method D used by the attacker is known as *distinguisher* in the SCA literature. Distinguishers studied in this study are explained in Section 3.5.4.
- The hypothesis with the best statistical score is the output of the attack: $\mathtt{K}_{[t']} \mid t' = \mathsf{argmax}_t(\mathsf{Score}(\mathtt{K}_{[t]}))$.

Notice that we treat X as a random variable defined as $X = \mathsf{G}(s, C)$, since it depends on the randomness of C. Also, above steps are simplified as if the attacker recorded a single power leakage for each execution of the attacked algorithm, which is typically not true. Usually, the attacker records multiple samples for each execution and takes the maximum during scoring: $\mathsf{Score}(\mathtt{K}_{[t]}) = \mathsf{max}_j(\mathsf{D}(L_j, X_t))$. In the literature, the set of leakage samples used in the attack is referred to as the point-of-interest (PoI).

At times, we use X' to denote the hypothetical intermediate value, and $\mathtt{X}'$ to denote the corresponding set of hypothetical intermediate values, in order to simplify the notation.

Example: We illustrate the application of the non-profiled attack framework described above as well as each distinguisher presented in the next section using a toy running example. This example further serves to demonstrate the masking countermeasure and the corresponding attacks against it.

Consider a toy cryptosystem where the partial secret key s is a 2-bit integer. Suppose an intermediate variable X in this scheme is computed as:

$$X = (C \mod 4) \cdot s + (C/4) \pmod 4 \tag{3.46}$$

In this example, 20 artificial traces were generated by simulating the power leakage.

The simulated traces are shown in Table 3.5. Each of the 20 traces contains four simulated leakage samples: one sample corresponds to the intermediate variable X as defined above, while the remaining samples correspond to random data. All samples were generated according to Equation (3.45) using the Hamming Weight leakage model $\mathsf{L} = \mathsf{HW}_2$, with parameters $\mu = 8$ and $\sigma = 0.25$. The trace generation procedure described here follows the approach outlined in Section 4.7, adapted for the toy encryption scheme.

Table 3.5 Simulated traces for the toy encryption scheme.

Trace Index (i)	$\mathtt{C}_{[i]}$	$\mathtt{L}_{0[i]}$	$\mathtt{L}_{1[i]}$	$\mathtt{L}_{2[i]}$	$\mathtt{L}_{3[i]}$
0	0	9.282	9.148	8.109	9.424
1	2	8.237	10.050	9.121	8.334
2	5	9.221	8.803	8.502	9.108
3	14	8.350	8.471	9.394	9.245
4	2	10.048	9.927	9.082	9.899
5	15	9.144	8.985	7.902	9.169
6	5	8.888	8.976	8.296	7.982
7	12	7.940	9.819	10.075	9.300
8	15	8.161	8.977	8.010	8.441
9	12	7.961	8.992	9.644	7.972
10	6	9.154	10.269	9.595	10.189
11	15	8.206	9.215	8.096	8.829
12	7	8.137	9.902	8.990	8.870
13	10	8.925	9.171	7.732	9.650
14	5	8.932	8.711	7.793	8.998
15	10	9.821	7.999	8.217	9.000
16	15	9.656	8.598	7.999	9.443
17	1	10.140	8.210	9.912	10.020
18	0	9.160	8.960	7.720	9.227
19	6	9.089	9.622	9.892	9.492

In this example, the secret key s was randomly selected as 3, and the leakage sample associated with the intermediate variable X was randomly selected as 2.

Since s is a 2-bit value, the set of hypotheses for s is $\mathtt{K} = \{0, 1, 2, 3\}$. Table 3.6 shows the computed sets of hypothetical intermediate values $\mathtt{X}_t$ for each hypothesis $\mathtt{K}_{[t]}$ across all traces.

Table 3.6 Hypothetical intermediate variables $\mathtt{X}_t$ for each hypothesis $\mathtt{K}_{[t]}$ in $\mathtt{K} = \{0,1,2,3\}$ in the toy example.

	$\mathtt{X}_{t[i]} = \mathsf{G}(\mathtt{K}_{[t]}, \mathtt{C}_{[i]})$			
i \ t	0	1	2	3
0	0	0	0	0
1	0	2	0	2
2	1	2	3	0
3	3	1	3	1
4	0	2	0	2
5	3	2	1	0
6	1	2	3	0
7	3	3	3	3
8	3	2	1	0
9	3	3	3	3
10	1	3	1	3
11	3	2	1	0
12	1	0	3	2
13	2	0	2	0
14	1	2	3	0
15	2	0	2	0
16	3	2	1	0
17	0	1	2	3
18	0	0	0	0
19	1	3	1	3

3.5.4. SCA Distinguishers

3.5.4.1. Correlation Power Analysis

Correlation Power Analysis (CPA) (Brier et al., 2004) is a widely-used side-channel distinguisher, based on Pearson's correlation. The attacker estimates the correlation coefficient between X' and L as follows:

$$\hat{\rho}(\mathsf{L}'(X'), L) = \frac{\hat{\mathsf{cov}}(\mathsf{L}'(\mathtt{X}'), \mathtt{L})}{\hat{\mathsf{std}}(\mathsf{L}'(\mathtt{X}')) \cdot \hat{\mathsf{std}}(\mathtt{L})} \tag{3.47}$$

CPA requires the attacker to predict the device's leakage function, denoted by L'. The most common choices for L' are the Hamming Weight function (HW) and the Hamming Distance function (HD), which model the state transitions in CMOS circuits.

Example: The computed hypothetical leakages $\mathsf{L}'(\mathtt{X}_t)$ for the running example are presented in Table 3.7 for each hypothesis $\mathtt{K}_{[t]}$, using $\mathsf{L}' = \mathsf{HW}_2$, along with

their standard deviations $\hat{\mathsf{std}}(\mathsf{HW}_2(\mathtt{X}_t))$. Table 3.8 shows the estimated covariance $\hat{\mathsf{cov}}(\mathtt{X}_t, \mathtt{L}_j)$ for each $\mathtt{K}_{[t]}$ and sample index j, with the standard deviation of each sample $\hat{\mathsf{std}}(\mathtt{L}_j)$ shown in the last column. The estimated correlation coefficients $\hat{\rho}(\mathsf{HW}_2(\mathtt{X}_t), L_j)$ for each hypothesis and sample are summarized in Table 3.9. Observe that the correlation coefficient for the correct hypothesis $\mathtt{K}_{[3]}$ and $j = 2$ is significantly higher than the others, demonstrating that CPA successfully recovers the secret key in this running example.

Table 3.7 Hypothetical leakages $\mathsf{L}'(\mathtt{X}_t)$ for each hypothesis $\mathtt{K}_t$ and the standard deviation of the hypothetical leakages in the toy example.

	$\mathsf{L}'(\mathsf{G}(\mathtt{K}_{[t]}, \mathtt{C}_{[i]})) = \mathsf{HW}_2(\mathtt{X}_{t[i]})$			
i \ t	0	1	2	3
0	0	0	0	0
1	0	1	0	1
2	1	1	2	0
3	2	1	2	1
4	0	1	0	1
5	2	1	1	0
6	1	1	2	0
7	2	2	2	2
8	2	1	1	0
9	2	2	2	2
10	1	2	1	2
11	2	1	1	0
12	1	0	2	1
13	1	0	1	0
14	1	1	2	0
15	1	0	1	0
16	2	1	1	0
17	0	1	1	2
18	0	0	0	0
19	1	2	1	2
$\hat{\mathsf{std}}(\mathsf{HW}_2(\mathtt{X}_t))$	0.788	0.686	0.745	0.865

Table 3.8 Estimated covariance between the hypothetical leakages $\mathsf{HW}_2(\mathtt{X}_t)$ and the observed leakages $\mathtt{L}_j$ for each hypothesis $\mathtt{K}_{[t]}$ and sample index j in the toy example. The last column shows the estimated standard deviation of the observed leakages $\mathtt{L}_j$.

	$\hat{\mathsf{cov}}(\mathsf{HW}_2(\mathtt{X}_t), \mathtt{L}_j)$				
j \ t	0	1	2	3	$\hat{\mathsf{std}}(\mathtt{L}_j)$
0	-0.279	-0.119	-0.214	-0.083	0.684
1	-0.080	0.140	-0.096	0.222	0.621
2	-0.012	0.376	0.134	0.682	0.822
3	-0.145	-0.005	-0.147	0.131	0.611

Table 3.9 Estimated correlation coefficients $\hat{\rho}(\mathsf{HW}_2(\mathtt{X}_t), \mathtt{L}_j)$ between the hypothetical leakages $\mathsf{HW}_2(\mathtt{X}_t)$ and the observed leakages $\mathtt{L}_j$ for each hypothesis $\mathtt{K}_{[t]}$ and sample index j in the toy example.

	$\hat{\rho}(\mathsf{HW}_2(\mathtt{X}_t), \mathtt{L}_j)$			
j \ t	0	1	2	3
0	-0.517	-0.253	-0.420	-0.140
1	-0.163	0.329	-0.208	0.413
2	-0.019	0.666	0.219	**0.960**
3	-0.302	-0.011	-0.323	0.247
$\max_j(\lvert\hat{\rho}(\mathsf{HW}_2(\mathtt{X}_t), \mathtt{L}_j)\rvert)$	0.517	0.666	0.420	**0.960**

3.5.4.2. Mutual Information Analysis

Mutual Information Analysis (MIA) (Gierlichs et al., 2008) is a *generic* and information-theoretic side-channel distinguisher. Unlike CPA, a generic distinguisher such as MIA does not require the attacker to predict a specific device leakage function L'. Let $\mathsf{H}(L)$ denote the entropy of L, and let $\mathsf{H}(L|X')$ denote the conditional entropy for L and X'. The mutual information (MI) between L and X' is defined as

$$\mathsf{I}(X', L) = \mathsf{H}(L) - \mathsf{H}(L|X') \tag{3.48}$$

Verbally, the entropy in L not covered by X', namely $\mathsf{H}(L|X')$, is subtracted from $\mathsf{H}(L)$ to formulate the mutual information in between. The entropy $\mathsf{H}(L)$ is defined as:

$$\mathsf{H}(L) = -\sum_{l \in \mathtt{L}} \mathsf{P}(L = l) \log(\mathsf{P}(L = l)) \tag{3.49}$$

while the conditional entropy $\mathsf{H}(L|X')$ is defined as:

$$\mathsf{H}(L|X') = -\sum_{x \in \mathtt{X}'} \mathsf{P}(X' = x) \sum_{l \in \mathtt{L}} \mathsf{P}(L = l | X' = x') \log(\mathsf{P}(L = l | X' = x)) \tag{3.50}$$

Computing H requires the probability density functions of its input. In practice, the attacker estimates $\hat{\mathsf{H}}$ over observations $\mathtt{L}$ and $\mathtt{X}'$. A common practice to estimate probabilities over observations is to use the histogram method, which partitions the observed data into bins and estimates the probabilities of each bin.

A limitation of generic distinguishers is that they can't distinguish if the target function G is an injective function. For instance, the modular multiplication, which is involved in the base multiplication in NTT domain (see Equation (3.32)) is an injective function. A solution to address this challenge to apply bit-dropping to the output of the target function, so it is no longer injective. For example, if the output of G is 8 bits, the attacker can discard the most significant 4 bits, retaining only the lower 4 bits. This modification ensures that G is no longer injective, allowing generic distinguishers to be effectively applied.

Example: Let $\mathsf{Bin}_{\mathtt{E}}(\mathtt{L}_j)$ denote the function mapping observed leakage $\mathtt{L}_{j[i]}$ to bin indexes in the histogram method, with respect to the bin edges $\mathtt{E}$. We use $\mathtt{E} = \{7.5, 8.5, 9.5, 10.5\}$, leveraging the knowledge that $\mu = 8$; intuitively, $\mathtt{E} = \{\mu - 0.5, \mu - 0.5 + \mathsf{HW}_2(1), \mu - 0.5 + \mathsf{HW}_2(2), \mu - 0.5 + \mathsf{HW}_2(3)\}$. In practice, the attacker does not know μ and σ, and thus the bin edges are typically selected based on the observed data. Table 3.10 shows the estimated histogram of the observed leakages $\mathtt{L}_j$ for each sample index j in the running example. For example, the bin index for the sample $\mathtt{L}_{0[0]}$ is 1, since this value satisfies $8.5 \leq 9.282 < 9.5$. The estimated probabilities for each bin are also shown in Table 3.10. The last row presents the estimated entropy $\hat{\mathsf{H}}(\mathtt{L}_j)$ for each j, calculated from the estimated probabilities using Equation (3.49). Table 3.11 presents the estimated joint probabilities $\hat{\mathsf{P}}(\mathsf{Bin}_{\mathtt{E}}(\mathtt{L}_j), \mathtt{X}_t)$ between the observed leakages $\mathtt{L}_j$ and the hypothetical intermediate values $\mathtt{X}_t$ for each hypothesis $\mathtt{K}_{[t]}$ and sample index j. For example, for $j = 0$ and hypothesis $\mathtt{K}_{[t]} = 0$, the estimated probability of observing $\mathsf{Bin}_{\mathtt{E}}(L_j) = 0$ and $\mathtt{X}_t = 0$ is 0.05. Note that each 4×3 rectangle in the table sums to 1. Table 3.12 shows the corresponding conditional probabilities $\hat{\mathsf{P}}(\mathsf{Bin}_{\mathtt{E}}(\mathtt{L}_j) \mid \mathtt{X}_t)$. Each 3×1 row in these tables sums to 1. Table 3.13 presents the estimated conditional entropies, calculated using the joint and conditional probabilities together with Equation (3.50). Finally, Table 3.14 presents the estimated mutual information $\hat{\mathsf{I}}(\mathtt{X}_t, \mathtt{L}_j)$, calculated using the entropy estimates for each $\mathtt{K}_{[t]}$ and j. Observe that the mutual information corresponding to the correct hypothesis $\mathtt{K}_{[3]}$ at $j = 2$ is substantially higher than the others, illustrating that MIA effectively recovers the secret key in this example.

Table 3.10 Histograms of the observed leakages L_j for each sample index j and the estimated probabilities in the toy example. The last row shows the estimated entropy $\hat{\mathsf{H}}(\mathsf{L}_j)$ for each j.

	$\mathsf{Bin}_{\mathbb{E}}(\mathsf{L}_{j[i]})$			
i \ j	0	1	2	3
0	1	1	0	1
1	0	2	1	0
2	1	1	1	1
3	0	0	1	1
4	2	2	1	2
5	1	1	0	1
6	1	1	0	0
7	0	2	2	1
8	0	1	0	0
9	0	1	2	0
10	1	2	2	2
11	0	1	0	1
12	0	2	1	1
13	1	1	0	2
14	1	1	0	1
15	2	0	0	1
16	2	1	0	1
17	2	0	2	2
18	1	1	0	1
19	1	2	2	1
b	$\hat{\mathsf{P}}(\mathsf{Bin}_{\mathbb{E}}(\mathsf{L}_j) = b)$			
0	0.35	0.15	0.50	0.20
1	0.45	0.55	0.25	0.60
2	0.20	0.30	0.25	0.20
$\hat{\mathsf{H}}(\mathsf{L}_j)$	1.05	0.97	1.04	0.95

Table 3.11 Estimated joint probability distributions $\hat{\mathsf{P}}(\mathsf{Bin}_{\mathbb{E}}(\mathsf{L}_j) = b, \mathsf{X}_t = x)$ for each hypothesis $\mathsf{K}_{[t]}$ and sample index j in the toy example.

		$\hat{\mathsf{P}}(\mathsf{Bin}_{\mathbb{E}}(\mathsf{L}_j) = b, \mathsf{X}_t = x)$											
j		0			1			2			3		
b		0	1	2	0	1	2	0	1	2	0	1	2
t	x												
0	0	0.05	0.10	0.10	0.05	0.10	0.10	0.10	0.10	0.05	0.05	0.10	0.10
0	1	0.05	0.25	0.00	0.00	0.15	0.15	0.10	0.10	0.10	0.05	0.20	0.05
0	2	0.00	0.05	0.05	0.05	0.05	0.00	0.10	0.00	0.00	0.00	0.05	0.05
0	3	0.25	0.05	0.05	0.05	0.25	0.05	0.20	0.05	0.10	0.10	0.25	0.00
1	0	0.05	0.15	0.05	0.05	0.15	0.05	0.20	0.05	0.00	0.00	0.20	0.05
1	1	0.05	0.00	0.05	0.10	0.00	0.00	0.00	0.05	0.05	0.00	0.05	0.05
1	2	0.15	0.20	0.10	0.00	0.35	0.10	0.30	0.15	0.00	0.15	0.25	0.05
1	3	0.10	0.10	0.00	0.00	0.05	0.15	0.00	0.00	0.20	0.05	0.10	0.05
2	0	0.05	0.10	0.05	0.00	0.10	0.10	0.10	0.10	0.00	0.05	0.10	0.05
2	1	0.10	0.15	0.05	0.00	0.20	0.10	0.20	0.00	0.10	0.05	0.20	0.05
2	2	0.00	0.05	0.10	0.10	0.05	0.00	0.10	0.00	0.05	0.00	0.05	0.10
2	3	0.20	0.15	0.00	0.05	0.20	0.10	0.10	0.15	0.10	0.10	0.25	0.00
3	0	0.10	0.35	0.10	0.05	0.50	0.00	0.50	0.05	0.00	0.10	0.40	0.05
3	1	0.05	0.00	0.00	0.05	0.00	0.00	0.00	0.05	0.00	0.00	0.05	0.00
3	2	0.10	0.00	0.05	0.00	0.00	0.15	0.00	0.15	0.00	0.05	0.05	0.05
3	3	0.10	0.10	0.05	0.05	0.05	0.15	0.00	0.00	0.25	0.05	0.10	0.10

Table 3.12 Estimated conditional probability distributions $\hat{\mathsf{P}}(\mathsf{Bin}_{\mathtt{E}}(\mathtt{L}_j) = b \mid \mathtt{X}_t = x)$ for each hypothesis $\mathtt{K}_{[t]}$ and sample index j in the toy example.

		$\hat{\mathsf{P}}(\mathsf{Bin}_{\mathtt{E}}(\mathtt{L}_j) = b \mid \mathtt{X}_t = x)$											
j		0			1			2			3		
b		0	1	2	0	1	2	0	1	2	0	1	2
t	x												
0	0	0.20	0.40	0.40	0.20	0.40	0.40	0.40	0.40	0.20	0.20	0.40	0.40
0	1	0.17	0.83	0.00	0.00	0.50	0.50	0.33	0.33	0.33	0.17	0.67	0.17
0	2	0.00	0.50	0.50	0.50	0.50	0.00	1.00	0.00	0.00	0.00	0.50	0.50
0	3	0.71	0.14	0.14	0.14	0.71	0.14	0.57	0.14	0.29	0.29	0.71	0.00
1	0	0.20	0.60	0.20	0.20	0.60	0.20	0.80	0.20	0.00	0.00	0.80	0.20
1	1	0.50	0.00	0.50	1.00	0.00	0.00	0.00	0.50	0.50	0.00	0.50	0.50
1	2	0.33	0.44	0.22	0.00	0.78	0.22	0.67	0.33	0.00	0.33	0.56	0.11
1	3	0.50	0.50	0.00	0.00	0.25	0.75	0.00	0.00	1.00	0.25	0.50	0.25
2	0	0.25	0.50	0.25	0.00	0.50	0.50	0.50	0.50	0.00	0.25	0.50	0.25
2	1	0.33	0.50	0.17	0.00	0.67	0.33	0.67	0.00	0.33	0.17	0.67	0.17
2	2	0.00	0.33	0.67	0.67	0.33	0.00	0.67	0.00	0.33	0.00	0.33	0.67
2	3	0.57	0.43	0.00	0.14	0.57	0.29	0.29	0.43	0.29	0.29	0.71	0.00
3	0	0.18	0.64	0.18	0.09	0.91	0.00	0.91	0.09	0.00	0.18	0.73	0.09
3	1	1.00	0.00	0.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00
3	2	0.67	0.00	0.33	0.00	0.00	1.00	0.00	1.00	0.00	0.33	0.33	0.33
3	3	0.40	0.40	0.20	0.20	0.20	0.60	0.00	0.00	1.00	0.20	0.40	0.40

Table 3.13 Estimated conditional entropy $\hat{\mathsf{H}}(\mathtt{L}_j \mid \mathtt{X}_t)$ for each hypothesis $\mathtt{K}_{[t]}$ and sample index j in the toy example.

	$\hat{\mathsf{H}}(\mathtt{L}_j \mid \mathtt{X}_t)$			
j \ t	0	1	2	3
0	0.75	0.92	0.85	0.86
1	0.82	0.59	0.76	0.41
2	0.93	0.48	0.80	0.17
3	0.80	0.82	0.77	0.85

Table 3.14 Estimated mutual information values $\hat{\mathsf{I}}(\mathtt{X}_t, \mathtt{L}_j)$ for each hypothesis $\mathtt{K}_{[t]}$ and sample index j in the toy example.

	$\hat{\mathsf{I}}(\mathtt{X}_t, \mathtt{L}_j)$			
j \ t	0	1	2	3
0	0.30	0.13	0.20	0.19
1	0.15	0.39	0.22	0.57
2	0.11	0.56	0.24	**0.87**
3	0.15	0.13	0.18	0.10
$\mathsf{max}_j(\hat{\mathsf{I}}(\mathtt{X}_t, \mathtt{L}_j))$	0.30	0.56	0.24	**0.87**

3.5.4.3. Kruskal-Wallis test

Kruskal-Wallis (KW) test (Yan et al., 2023) is another generic distinguisher based on ranked side-channel leakages. Initially, the observed leakages $\mathtt{L}$ are ranked from smallest to largest while the ties are resolved by assigning the average of the ranks that the ties would have received, denoted by $\mathtt{R} = \mathsf{rank}(\mathtt{L})$. The ranked data are partitioned based on hypothetical intermediates $\mathtt{X}'$:

$$\mathtt{R}_x = \{\mathtt{R}_{[i]} \mid \mathtt{X'}_{[i]} = x, 0 \leq i < \nu\} \tag{3.51}$$

which contains the ranks of leakages corresponding the traces where the intermediate $X' = x$.

Then, KW statistic is defined as

$$\mathsf{KW}(\mathtt{X}', \mathtt{R}) = (\nu - 1)\frac{\sum_{x \in \mathtt{X}'} \nu_x (\overline{\mathtt{R}}_x - (\nu+1)/2)^2}{\sum_{x \in \mathtt{X}'} \sum_{i=0}^{\nu_x} (\mathtt{R}_{x[i]} - (\nu+1)/2)^2} \tag{3.52}$$

where ν_x denotes the number of elements in the set $\mathtt{R}_x$ and $\overline{\mathtt{R}}_x = (\sum_{i=0}^{\nu_x} \mathtt{R}_{x[i]})/\nu_x$.

Example: Table 3.15 shows the ranks of the observed leakages $\mathtt{L}_j$ for each sample index j in the running example. We explicitly include the sample index j in the notation to distinguish the ranks, denoted by $\mathtt{R}^j$. For example, the rank $\mathtt{R}^0_{[7]}$ is 0 since $\mathtt{L}_{0[7]}$ is the smallest among $\mathtt{L}_0$ in Table 3.5. Table 3.16 shows the partitioning of the traces with respect to the hypothetical intermediates $\mathtt{X}_t$, for each hypothesis $\mathtt{K}_{[t]}$. As an example, consider the partition corresponding to hypothesis $\mathtt{K}_{[0]} = 0$ and hypothetical intermediate value $\mathtt{X}_t = 0$. In this case, the traces $\{0, 1, 4, 17, 18\}$ are grouped together (also see Table 3.6). Notice that number of partitions is 4 since the hypothetical values $\mathtt{X}_t$ take values $x \in \{0, 1, 2, 3\}$. This partitioning is necessary to construct the sets of ranks $\mathtt{R}^{j,t}_x$, as shown in Table 3.17 for each x, j, and $\mathtt{K}_{[t]}$. Note that the notation explicitly includes the hypothesis index t and sample index j for clarity. The partitions are constructed by mapping the trace indices from Table 3.16 to their corresponding rank values for each j. The average ranks $\overline{\mathtt{R}}^{j,t}_x$ for each partition are summarized in Table 3.18. Finally, Table 3.19 presents the computed KW statistics $\mathsf{KW}(\mathtt{X}_t, \mathtt{R})$ for each hypothesis $\mathtt{K}_{[t]}$ and sample index j, calculated using Equation (3.52) and the partitioned ranks $\mathtt{R}^{j,t}_x$. Observe that the KW statistic for the correct hypothesis $\mathtt{K}_{[3]}$ is significantly higher than the others, demonstrating that KW successfully recovers the secret key in this running example.

Table 3.15 Ranks $\mathtt{R}^j$ of the observed leakages $\mathtt{L}_j$ for each sample index j in the toy example.

i \ j	$\mathtt{R}^j_{[i]} \mid \mathtt{R}^j = \mathsf{rank}(\mathtt{L}_j)$			
	0	1	2	3
0	15	11	7	13
1	5	18	13	2
2	14	5	10	8
3	6	2	14	11
4	18	17	12	17
5	11	9	3	9
6	7	7	9	1
7	0	15	19	12
8	3	8	5	3
9	1	10	16	0
10	12	19	15	19
11	4	13	6	4
12	2	16	11	5
13	8	12	1	16
14	9	4	2	6
15	17	0	8	7
16	16	3	4	14
17	19	1	18	18
18	13	6	0	10
19	10	14	17	15

Table 3.16 Partitioning of the traces with respect to the hypothetical intermediates $\mathtt{X}_t$ for each hypothesis $\mathtt{K}_{[t]}$ in the toy example.

x \ t	$\{i \mid \mathtt{X}_{t[i]} = x, 0 \leq i < \nu\}$			
	0	1	2	3
0	0, 1, 4, 17, 18	0, 12, 13, 15, 18	0, 1, 4, 18	0, 2, 5, 6, 8, 11, 13, 14, 15, 16, 18
1	2, 6, 10, 12, 14, 19	3, 17	5, 8, 10, 11, 16, 19	3
2	13, 15	1, 2, 4, 5, 6, 8, 11, 14, 16	13, 15, 17	1, 4, 12
3	3, 5, 7, 8, 9, 11, 16	7, 9, 10, 19	2, 3, 6, 7, 9, 12, 14	7, 9, 10, 17, 19

Table 3.17 Partitioned ranks $\mathtt{R}_x^{j,t}$ with respect to hypothetical intermediates x for each hypothesis $\mathtt{K}_{[t]}$ and sample index j in the toy example.

		$\mathtt{R}_x^{j,t} = \{\mathtt{R}_{[i]}^{j} \mid \mathtt{X}_{t[i]} = x, 0 \leq i < \nu\} \mid \mathtt{R}^j = \mathsf{rank}(\mathtt{L}_j)$			
j		0	1	2	3
t	x				
0	0	15, 5, 18, 19, 13	11, 18, 17, 1, 6	7, 13, 12, 18, 0	13, 2, 17, 18, 10
0	1	14, 7, 12, 2, 9, 10	5, 7, 19, 16, 4, 14	10, 9, 15, 11, 2, 17	8, 1, 19, 5, 6, 15
0	2	8, 17	12, 0	1, 8	16, 7
0	3	6, 11, 0, 3, 1, 4, 16	2, 9, 15, 8, 10, 13, 3	14, 3, 19, 5, 16, 6, 4	11, 9, 12, 3, 0, 4, 14
1	0	15, 2, 8, 17, 13	11, 16, 12, 0, 6	7, 11, 1, 8, 0	13, 5, 16, 7, 10
1	1	6, 19	2, 1	14, 18	11, 18
1	2	5, 14, 18, 11, 7, 3, 4, 9, 16	18, 5, 17, 9, 7, 8, 13, 4, 3	13, 10, 12, 3, 9, 5, 6, 2, 4	2, 8, 17, 9, 1, 3, 4, 6, 14
1	3	0, 1, 12, 10	15, 10, 19, 14	19, 16, 15, 17	12, 0, 19, 15
2	0	15, 5, 18, 13	11, 18, 17, 6	7, 13, 12, 0	13, 2, 17, 10
2	1	11, 3, 12, 4, 16, 10	9, 8, 19, 13, 3, 14	3, 5, 15, 6, 4, 17	9, 3, 19, 4, 14, 15
2	2	8, 17, 19	12, 0, 1	1, 8, 18	16, 7, 18
2	3	14, 6, 7, 0, 1, 2, 9	5, 2, 7, 15, 10, 16, 4	10, 14, 9, 19, 16, 11, 2	8, 11, 1, 12, 0, 5, 6
3	0	15, 14, 11, 7, 3, 4, 8, 9, 17, 16, 13	11, 5, 9, 7, 8, 13, 12, 4, 0, 3, 6	7, 10, 3, 9, 5, 6, 1, 2, 8, 4, 0	13, 8, 9, 1, 3, 4, 16, 6, 7, 14, 10
3	1	6	2	14	11
3	2	5, 18, 2	18, 17, 16	13, 12, 11	2, 17, 5
3	3	0, 1, 12, 19, 10	15, 10, 19, 1, 14	19, 16, 15, 18, 17	12, 0, 19, 18, 15

Table 3.18 Average ranks $\overline{\mathtt{R}}_x^{j,t}$ with respect to hypothetical intermediates x for each $\mathtt{K}_{[t]}$ and j in the toy example.

		$\overline{\mathtt{R}}_x^{j,t} \mid \mathtt{R}_x^{j,t} = \{\mathtt{R}_{[i]}^j \mid \mathtt{X}_{t[i]} = x, 0 \leq i < \nu\}, \mathtt{R}^j = \mathsf{rank}(\mathtt{L}_j)$			
j		0	1	2	3
t	x				
0	0	14.00	10.60	10.00	12.00
0	1	9.00	10.83	10.67	9.00
0	2	12.50	6.00	4.50	11.50
0	3	5.86	8.57	9.57	7.57
1	0	11.00	9.00	5.40	10.20
1	1	12.50	1.50	16.00	14.50
1	2	9.67	9.33	7.11	7.11
1	3	5.75	14.50	16.75	11.50
2	0	12.75	13.00	8.00	10.50
2	1	9.33	11.00	8.33	10.67
2	2	14.67	4.33	9.00	13.67
2	3	5.57	8.43	11.57	6.14
3	0	10.64	7.09	5.00	8.27
3	1	6.00	2.00	14.00	11.00
3	2	8.33	17.00	12.00	8.00
3	3	8.40	11.80	17.00	12.80

Table 3.19 $\mathsf{KW}(\mathtt{X}_t, \mathtt{L}_j)$ statistics for each hypothesis $\mathtt{K}_{[t]}$ and sample index j in the toy example.

	$\mathsf{KW}(\mathtt{X}_t, \mathtt{L}_j)$			
j \ t	0	1	2	3
0	227662.469	215077.797	229324.719	210240.250
1	211289.641	229222.797	221460.922	237664.609
2	212490.125	248967.859	211358.531	**260071.203**
3	213211.719	218429.406	220725.281	214513.781
$\mathsf{max}_j(\mathsf{KW}(\mathtt{X}_t, \mathtt{L}_j))$	227662.469	248967.859	229324.719	**260071.203**

3.5.5. Masking Countermeasure

Masking (Chari et al., 1999) is the state-of-the-art countermeasure against power leakage based SCA attacks. The masking countermeasure operates as a secret sharing scheme through the cryptographic execution. The level of protection is determined by the masking order. Specifically, a $(d-1)$-th order masking scheme utilizes d shares to provide resistance against $d-1$ simultaneous leakages.

Various types of masking exist in the literature, which are distinguished by the operator used to combine the shares. The most frequently employed masking schemes are *boolean* and *arithmetic* masking. In boolean masking, for a sensitive variable X, the shares $X^{\{0\}}, X^{\{1\}}, \ldots, X^{\{d-1\}}$ satisfy:

$$X = X^{\{0\}} \bigoplus X^{\{1\}} \bigoplus \cdots \bigoplus X^{\{d-1\}} \tag{3.53}$$

where $\bigoplus$ denotes the bit-wise exclusive-or operation.

On the other hand, in arithmetic masking, the secret value is shared as:

$$X = X^{\{0\}} + X^{\{1\}} + \cdots + X^{\{d-1\}} \pmod{q} \tag{3.54}$$

Masking is effective because the power leakages corresponding to any $d-1$ shares, *e.g.*, $\{\mathsf{L}(X^{\{i\}})\}_{i=0}^{d-2}$, are statistically independent of the sensitive intermediate value X.

In masked implementations, sensitive variables such as secret keys, nonces, and seeds are split into d shares as described above at the start of execution. The entire algorithm is then executed independently over these shares. Specifically, to mask a secret s, the shares $S^{\{0\}}, \ldots, S^{\{d-2\}}$ are sampled uniformly at random, and the final share is computed as $S^{\{d-1\}} = s \bigoplus (\bigoplus_{i=0}^{d-2} S^{\{i\}})$ for boolean masking, or $S^{\{d-1\}} = s - \sum_{i=0}^{d-2} S^{\{i\}} \pmod{q}$ for arithmetic masking. Throughout execution, the shares of intermediate variables maintain the secret sharing relationship defined by Equation (3.53), Equation (3.54), or any other relevant secret sharing relation, depending on the masking scheme employed.

Running the algorithm independently over the shares is straightforward for operations that are *linear* with respect to the masking scheme. For example, boolean operations such as XOR and bit shifts are compatible with boolean masking, while arithmetic operations such as addition and multiplication are compatible with arithmetic masking. However, masking *non-linear* operations presents significant challenges and has been the subject of extensive research. Examples of non-linear opera-

tions in boolean masking include the AES S-Box (Canright & Batina, 2008) and the chi transformation in Keccak (Groß, Schaffenrath & Mangard, 2017). In arithmetic masking, non-linear operations such as polynomial compression and decomposition in LBC are particularly challenging to mask efficiently and securely (Coron, Gérard, Trannoy & Zeitoun, 2023; D'Anvers, Van Beirendonck & Verbauwhede, 2022).

Example:

To illustrate masking in the context of the running toy encryption scheme, consider arithmetic masking with two shares ($d = 2$). The 2-bit secret s is split into two shares, $S^{\{0\}}$ and $S^{\{1\}}$, such that $s = S^{\{0\}} + S^{\{1\}} \pmod 4$. The computation of the intermediate variable X can then be expressed as:

$$X^{\{0\}} = (C \mod 4) \cdot S^{\{0\}} + (C/4) \pmod 4 \tag{3.55}$$

$$X^{\{1\}} = (C \mod 4) \cdot S^{\{1\}} \pmod 4 \tag{3.56}$$

Notice that $X^{\{0\}} + X^{\{1\}} = (C \mod 4) \cdot s + (C/4) \pmod 4$ aligning with Equation (3.46).

Table 3.20 presents the simulated traces for the masked execution of the toy encryption scheme, where the intermediate variable X is computed from the shares $X^{\{0\}}$ and $X^{\{1\}}$ as described above. The secret shares $S^{\{0:1\}}$ are also demonstrated for each trace. In this setting, each trace contains eight samples: two correspond to the leakages of $X^{\{0\}}$ and $X^{\{1\}}$, while the remaining samples represent simulated leakages from unrelated random data. Observe from Table 3.20 that the *first-order* CPA can't distinguish the correct key from the others, as the leakages of $X^{\{0\}}$ and $X^{\{1\}}$ are statistically independent of the secret key s.

Table 3.20 Simulated traces for the toy encryption scheme with masking.

Trace Index (i)	$\mathtt{C}_{[i]}$	$S_{[i]}^{\{0\}}$	$S_{[i]}^{\{1\}}$	$\mathtt{L}_{0[i]}$	$\mathtt{L}_{1[i]}$	$\mathtt{L}_{2[i]}$	$\mathtt{L}_{3[i]}$	$\mathtt{L}_{4[i]}$	$\mathtt{L}_{5[i]}$	$\mathtt{L}_{6[i]}$	$\mathtt{L}_{7[i]}$
0	0	1	2	9.385	10.425	8.026	8.006	8.809	9.260	8.237	8.050
1	10	0	3	8.753	9.249	9.484	8.880	8.954	7.923	9.191	9.314
2	13	1	2	7.933	10.351	8.234	7.755	10.183	7.954	9.041	10.226
3	13	0	3	8.886	9.762	9.884	8.607	9.798	7.755	9.644	9.116
4	12	2	1	9.010	9.441	9.954	7.996	9.166	8.889	8.222	7.783
5	2	3	0	8.154	9.269	8.595	10.189	8.804	9.847	8.101	10.213
6	0	3	0	9.199	7.790	8.137	9.902	7.990	7.870	7.728	7.597
7	13	1	2	7.907	10.384	8.109	8.347	8.932	8.711	8.793	8.998
8	10	0	3	8.112	9.446	8.828	9.908	9.109	10.198	9.656	8.598
9	9	3	0	9.988	8.334	9.336	10.248	8.039	10.071	7.980	7.690
10	5	1	2	9.219	8.737	8.958	10.166	10.302	7.743	9.086	7.914
11	9	2	1	9.088	9.156	7.998	9.965	10.304	10.078	8.863	10.114
12	15	3	0	8.428	8.860	7.961	9.129	10.391	8.843	7.811	10.477
13	3	0	3	8.186	8.030	7.828	9.005	8.011	8.097	9.001	9.954
14	1	2	1	8.554	10.286	8.718	7.861	10.044	9.711	9.291	9.189
15	0	2	1	9.965	8.152	7.915	7.719	9.154	7.769	7.885	8.835
16	14	0	3	8.834	7.880	9.830	7.921	8.502	8.779	8.915	9.267
17	13	0	3	9.874	8.718	10.174	8.105	8.096	8.104	9.827	7.981
18	6	3	0	8.067	9.266	9.962	8.409	9.061	10.845	7.823	10.443
19	11	1	2	9.374	10.207	9.036	8.602	8.855	10.060	9.229	10.230

Table 3.21 Estimated correlation coefficients $\hat{\rho}(\mathsf{HW}_2(\mathtt{X}_t), \mathtt{L}_j)$ between the hypothetical leakages $\mathsf{HW}_2(\mathtt{X}_t)$ and the observed leakages $\mathtt{L}_j$ for each hypothesis $\mathtt{K}_{[t]}$ and sample index j in the toy example with masking.

	$\hat{\rho}(\mathsf{HW}_2(\mathtt{X}_t), \mathtt{L}_j)$			
j \ t	0	1	2	3
0	-0.134	-0.027	-0.305	-0.160
1	0.165	-0.240	-0.046	0.333
2	0.477	0.130	0.474	0.426
3	-0.255	0.258	-0.265	-0.334
4	0.248	0.005	0.236	-0.018
5	-0.154	0.508	-0.288	0.466
6	0.307	-0.300	0.339	0.116
7	0.057	0.229	-0.103	0.291
$\max_j(\lvert\hat{\rho}(\mathsf{HW}_2(\mathtt{X}_t), \mathtt{L}_j)\rvert)$	0.477	0.508	0.474	0.466

3.5.6. Attacks on Masking

Next, we discuss how SCA distinguishers are extended to be effective in the presence of masking countermeasures. In general, attacking a $(d-1)$-th order masked implementation requires a d-th order SCA attack. A d-th order attack utilizes the leakages of d secret shares, denoted as $L_0, L_1, \ldots, L_{d-1}$.

3.5.6.1. Higher-order CPA

To perform a higher-order CPA (HOCPA) attack on a masked implementation, a function that combines the leakage of shares $L_0, L_1, .., L_{d-1}$ must be used. The most frequently used combination function is the mean-free product (Prouff et al., 2009), which is defined as follows for a d-th order attack:

$$\mathsf{C}(L_0, L_1, .., L_{d-1}) = \prod_{i=0}^{d-1} \Big(L_i - \mathsf{E}[L_i]\Big) \tag{3.57}$$

To simplify the notation, we use $\mathsf{C}(\{L_i\}_{i=0}^{d-1})$ for the above definition. A challenge here is to find the subset of samples to combine through the mean-free product or another combination function. In most circumstances, performing the combination exhaustively for all leakage samples may be infeasible.

To efficiently perform a HOCPA attack by means of a combination function, an optimal prediction function must be calculated (Prouff et al., 2009). Specifically, for a d-th order attack:

$$\mathsf{F}^d_{\text{opt}}(x) = \mathsf{E}\Big[\mathsf{C}\Big(\{\mathsf{L}'(X^{\{i\}})\}_{i=0}^{d-1}\Big)\Big|X = x\Big] \tag{3.58}$$

where $X = \sum_{i=0}^{d-1} X^{\{i\}}$ thus the expectation is taken over the random shares $X^{\{0:d-2\}}$. The attacker then estimates $\hat{\rho}(\mathsf{F}^d_{\text{opt}}(X'), \mathsf{C}(\{L_i\}_{i=0}^{d-1}))$. The correlation achieved by the optimal prediction function f^d_{opt} is called *optimal correlation* denoted by ρ_{opt}:

$$\rho_{opt} = \rho(\mathsf{F}^d_{\text{opt}}(X), \mathsf{C}(\{\mathsf{L}'(X^{\{i\}})\}_{i=0}^{d-1})) \tag{3.59}$$

In the case of boolean masking, it has been shown that $\mathsf{F}^d_{\text{opt}} = \mathsf{HW}_\beta$ for all d and β (Prouff et al., 2009). However, for arithmetic masking, deriving $\mathsf{F}^d_{\text{opt}}$ is generally more involved and may not have a simple closed-form expression. In such cases, the

optimal prediction function must often be computed numerically or approximated based on the statistical properties of the shares and the leakage function.

Another commonly used combination function is the absolute value combination function (Joye, Paillier & Schoenmakers, 2005), defined as follows for $d = 2$:

$$\mathsf{C}_{\text{abs}}\Big(\mathsf{L}(X^{\{0\}}), \mathsf{L}(X^{\{1\}})\Big) = \Big|\mathsf{L}(X^{\{0\}}) - \mathsf{L}(X^{\{1\}})\Big| \tag{3.60}$$

Note that we omit the subscript when referring to the mean-free product combination function C, whereas we explicitly include it when referring to the absolute value combination function C_{abs}. This is because the mean-free product is used much more frequently in this study.

Example: Table 3.22 illustrates the application of the mean-free product combination function C to the simulated traces from the toy example, as previously shown in Table 3.20. For clarity, not all possible combinations of the simulated samples are included. Since each trace contains 8 samples, there are 64 possible pairwise combinations. Here, we present only the combinations where the sample at index $j_0 = 0$ is combined with each sample at indices $0 \leq j_1 < 8$. In this setting, The optimal prediction function is defined (by considering $\mathsf{L}' = \mathsf{HW}_2$) as:

$$\mathsf{F}^2_{\text{opt}}(0) = 0.25,\ \mathsf{F}^2_{\text{opt}}(1) = 0,\ \mathsf{F}^2_{\text{opt}}(2) = 0.25,\ \mathsf{F}^2_{\text{opt}}(3) = -0.5 \tag{3.61}$$

For example, for $x = 0$:

$$\mathsf{F}^2_{\text{opt}}(0) = \frac{\sum_{i=0}^{3} \mathsf{C}(\mathsf{HW}_2(i), \mathsf{HW}_2(4-i \mod 4))}{4} \tag{3.62}$$

$$= \frac{\sum_{i=0}^{3} (\mathsf{HW}_2(i) - 1) \cdot (\mathsf{HW}_2((4-i \mod 4) - 1))}{4} \tag{3.63}$$

$$= \frac{(-1)\cdot(-1) + 0\cdot 1 + 0\cdot 01\cdot 0}{4} \tag{3.64}$$

$$= 0.25 \tag{3.65}$$

Notice that $\mathsf{E}[\mathsf{HW}_2(X)] = 1$ when X is uniformly distributed over $\{0,1,2,3\}$.

For brevity, we omit the detailed computations for HOCPA, as they follow the same procedure as CPA, with HW_2 replaced by $\mathsf{F}^2_{\text{opt}}$ and $\mathtt{L}_j$ replaced by $\mathsf{C}(\{\mathtt{L}_{j_0}, \mathtt{L}_{j_1}\})$.

Table 3.22 Combinations of the observed leakages in the toy example with masking. Leakages for sample index $j_0 = 0$ is combined to all $0 \leq j_1 < 8$.

	$\mathsf{C}(\mathtt{L}_{j_0}, \mathtt{L}_{j_1})$							
i \ j_1	0	1	2	3	4	5	6	7
0	0.291	0.667	-0.443	-0.447	-0.170	0.181	-0.258	-0.566
1	0.009	-0.006	-0.059	-0.004	0.016	0.093	-0.044	-0.020
2	0.832	-1.062	0.560	0.986	-0.965	0.886	-0.296	-1.028
3	0.002	0.023	0.041	-0.009	0.027	-0.047	0.037	0.001
4	0.027	0.042	0.182	-0.138	0.007	-0.006	-0.081	-0.217
5	0.479	-0.057	0.175	-0.936	0.223	-0.638	0.426	-0.771
6	0.125	-0.494	-0.252	0.377	-0.402	-0.373	-0.349	-0.531
7	0.882	-1.125	0.694	0.459	0.182	0.201	-0.072	0.095
8	0.539	-0.190	0.015	-0.787	0.012	-0.934	-0.690	0.368
9	1.305	-0.975	0.557	1.613	-1.241	1.309	-0.841	-1.610
10	0.140	-0.168	0.041	0.497	0.440	-0.442	0.138	-0.443
11	0.058	-0.008	-0.206	0.273	0.285	0.279	0.035	0.245
12	0.175	0.137	0.371	-0.123	-0.529	0.034	0.379	-0.576
13	0.435	0.763	0.673	-0.111	0.735	0.547	-0.188	-0.563
14	0.085	-0.321	0.038	0.284	-0.268	-0.229	-0.168	-0.026
15	1.253	-1.159	-1.044	-1.251	0.032	-1.295	-0.931	-0.295
16	0.000	0.016	-0.012	0.011	0.007	0.002	-0.002	-0.002
17	1.058	-0.482	1.364	-0.752	-1.058	-0.845	1.142	-1.150
18	0.607	-0.061	-0.867	0.332	0.050	-1.495	0.696	-1.046
19	0.279	0.538	0.099	-0.124	-0.143	0.599	0.271	0.597

3.5.6.2. Multivariate Mutual Information Analysis

MIA trivially generalizes to a higher-order attack, known as Multivariate Mutual Information Analysis (MMIA). For $d = 2$:

$$\mathsf{I}(L_0, L_1, X') = \mathsf{I}(L_0, L_1) - \mathsf{I}(L_0, L_1 | X') \tag{3.66}$$

the conditionally mutual information is defined as:

$$\mathsf{I}(L_0, L_1 \mid X') = \sum_{x \in \mathtt{X}'} \mathsf{P}(X' = x) \cdot \mathsf{I}(L_0, L_1 \mid X = x) \tag{3.67}$$

In this setting, the probability distributions of both L_0 and L_1 are estimated.

Example: Table 3.23 shows the histogram of the observed leakages $\mathtt{L}_j$ for each sample index j in the masked toy example. We omit the detailed calculations for MMIA due to their complexity. However, the process is a straightforward extension of the MIA procedure, applying the above formulation and using the histograms of two leakage samples.

Table 3.23 Histograms of the observed leakages $\mathtt{L}_j$ for each sample index j and the estimated probabilities in the toy example with masking.

	$\mathsf{Bin}_{\mathbb{E}}(\mathtt{L}_{j[i]})$							
i \ j	0	1	2	3	4	5	6	7
0	10	18	2	4	6	13	5	6
1	2	12	5	10	11	12	10	13
2	15	2	16	18	2	18	4	3
3	1	14	10	9	12	8	13	15
4	3	15	13	5	8	9	8	11
5	12	10	12	1	16	4	17	4
6	6	4	3	15	4	6	3	8
7	16	1	18	16	15	14	9	16
8	13	7	7	2	10	2	2	18
9	19	3	15	19	0	19	1	0
10	7	8	9	17	18	5	14	9
11	4	11	4	12	17	15	12	17
12	8	16	14	7	3	11	16	5
13	11	19	17	8	19	16	6	7
14	5	6	8	13	5	7	7	12
15	18	0	0	0	13	1	0	10
16	0	13	6	11	9	10	11	14
17	17	5	19	3	1	3	19	1
18	14	9	1	14	14	0	18	2
19	9	17	11	6	7	17	15	19

3.5.6.3. Higher-order KW

Like CPA, a higher-order attack using KW is performed using a combination function such as C. The attacker computes the ranks of the combined leakages $\mathsf{C}(\{L_i\}_{i=0}^{d-1})$:

$$\mathtt{R} = \mathsf{rank}(\mathsf{C}(\{L_i\}_{i=0}^{d-1})) \tag{3.68}$$

The remainder of the procedure follows as detailed in Section 3.5.4.3.

Example: As an illustrative example, Table 3.24 presents the ranks of the combined leakages for the masked toy example. Note that we show only the ranks corresponding to combinations where the sample at index $j_0 = 0$ is combined with each sample at indices $0 \leq j_1 < 8$. We skip the subsequent computations for the

second-order KW since they follow exactly the same procedure as the first-order KW described in Section 3.5.4.3, but are applied to the ranks of the combined leakages.

Table 3.24 Ranks $\mathtt{R}^{j_0,j_1}$ of the combinations of the observed leakages $\mathsf{C}(\mathsf{L}_{j_0},\mathsf{L}_{j_1})$ in the toy example with masking. Leakages for sample index $j_0 = 0$ is combined to all $0 \leq j_1 < 8$.

	$\mathtt{R}^{j_0,j_1}_{[i]} \mid \mathtt{R}^{j_0,j_1} = \mathsf{rank}(\mathsf{C}(\mathsf{L}_{j_0},\mathsf{L}_{j_1}))$							
i \ j	0	1	2	3	4	5	6	7
0	11	19	3	5	7	14	6	7
1	3	13	6	11	12	13	11	14
2	16	3	17	19	3	19	5	4
3	2	15	11	10	13	9	14	16
4	4	16	14	6	9	10	9	12
5	13	11	13	2	17	5	18	5
6	7	5	4	16	5	7	4	9
7	17	2	19	17	16	15	10	17
8	14	8	8	3	11	3	3	19
9	20	4	16	20	1	20	2	1
10	8	9	10	18	19	6	15	10
11	5	12	5	13	18	16	13	18
12	9	17	15	8	4	12	17	6
13	12	20	18	9	20	17	7	8
14	6	7	9	14	6	8	8	13
15	19	1	1	1	14	2	1	11
16	1	14	7	12	10	11	12	15
17	18	6	20	4	2	4	20	2
18	15	10	2	15	15	1	19	3
19	10	18	12	7	8	18	16	20

4. NON-PROFILED SCA ON NTT MULTIPLICATION

In this chapter, we briefly describe how non-profiled SCA attacks are applied to NTT-based polynomial multiplication as used in Dilithium and Kyber.

4.1. Adversary Model

Recall that SCA attacks target cryptographic function executions that involve secret keys. In this context, we focus on the signature generation function in Dilithium (Algorithm 8), which involves secret vectors of polynomials $\mathbf{s_1}$ and $\mathbf{s_2}$. Likewise, we target the secret key decryption function in Kyber (Algorithm 6), which uses the secret polynomial vector $\mathbf{s}$.

Throughout this study, we denote the targeted secret polynomial in LBC by s, and its NTT domain representation by $\hat{s}$. This notation can refer to any individual polynomial $\mathbf{s}_{[i]}$ in Kyber, or $\mathbf{s_1}_{[i]}$ or $\mathbf{s_2}_{[i]}$ in Dilithium, for any index i. The described methodology is applied repeatedly to recover all polynomials within these secret vectors.

To perform a non-profiled SCA attack on s, a natural choice for the target operation is the NTT-based polynomial multiplication (Chen et al., 2021; Mujdei et al., 2024; Tosun et al., 2024,2; Tosun & Savas, 2024). In Kyber, this corresponds to one of the polynomial multiplications $\mathbf{u}_{[i]} \cdot \mathbf{s}_{[i]}$ (see line 3 of Algorithm 3) for some index i. Similarly, in Dilithium, it may refer to one of the polynomial multiplications $c \cdot \mathbf{s}_{1[i]}$ or $c \cdot \mathbf{s}_{2[i]}$ (see lines 12–13 and 17 of Algorithm 8).

We denote the publicly known operand involved in the NTT-based polynomial multiplication by c, and its NTT representation by $\hat{c}$. Specifically, in Kyber, c corresponds to any of the ciphertext polynomials $\mathbf{u}_{[i]}$, and in Dilithium, c refers to the challenge polynomial (also denoted as c in Algorithm 8). In both cases, c is publicly known

and varies across executions of the attacked function.

More precisely, the target of the attack is the base multiplication $\hat{s} \star \hat{c}$ performed in the NTT domain, which reduces to a set of independent products $\hat{s}_{[i]} \cdot \hat{c}_{[i]}$, enabling each $\hat{s}_{[i]}$ to be attacked individually. In the following two sections, we present the details of these attacks in the context of complete and incomplete NTTs, corresponding to Dilithium and Kyber, respectively.

4.2. Attack on Complete NTT

First, we describe the attack when the NTT is complete as in Dilithium. As previously mentioned, the base multiplication in the NTT domain is performed elementwise. Consequently, each coefficient $\hat{s}_{[i]}$ can be treated as a subkey and attacked independently using the corresponding public value $\hat{c}_{[i]}$. Under a complete NTT, this operation reduces to a set of independent modular multiplications of the form $\mathcal{G}(\hat{s}_{[i]}, \hat{c}_{[i]}) = \hat{s}_{[i]} \cdot \hat{c}_{[i]} \pmod{q}$ (see Equation (3.32)). Naturally, this results in a total of n independent attacks. To recover $\hat{s}_{[i]}$, the attacker tests q hypotheses. However, this number can be reduced to $q/2$ by leveraging the fact that the additive inverse of the correct secret coefficient also tends to produce statistically distinguishable scores during the SCA attack (Chen et al., 2021; Tosun et al., 2024,2; Tosun & Savas, 2024).

4.3. Attack on Incomplete NTT

For the $(\log(n) - 1)$-level incomplete NTT case as in Kyber and specific implementations of Dilithium (Abdulrahman et al., 2022), the base multiplication is presented in Equation (3.35)-Equation (3.36), which consists of independent multiplications of degree-1 polynomials. Consequently, the attacker performs $n/2$ independent attacks. The attacker can target

- The output lower degree-coefficient from these multiplications $\mathcal{G}(\hat{s}_{[i]}, \hat{c}_{[i]}) = \hat{r}_{[i][0]}$ (computed in Equation (3.35))

- The output higher degree coefficient $\mathcal{G}(\hat{s}_{[i]}, \hat{c}_{[i]}) = \hat{r}_{[i][1]}$ (computed in Equation (3.36))
- Or both $\mathcal{G}(\hat{s}_{[i]}, \hat{c}_{[i]}) = \hat{r}_{[i][0]} || \hat{r}_{[i][1]}$.

For any of these target functions, both $\hat{r}_{[i][0]}$ and $\hat{r}_{[i][1]}$ depend on both $\hat{s}_{[i][0]}$ and $\hat{s}_{[i][1]}$, so these must be predicted together (Mujdei et al., 2024; Tosun et al., 2024; Tosun & Savas, 2024). As a result, the attacker must evaluate q^2 hypotheses. Similar to the complete NTT case, this number can be reduced to $q^2/2$ by exploiting the fact that additive inverses of the tested hypotheses yield similarly distinguishable leakage characteristics.

4.4. Lattice Attacks

Prior research has demonstrated that recovering a subset of the NTT domain secret coefficients $\hat{s}_i$ through the SCA attack is sufficient to efficiently compute the entire normal domain secret polynomial s (Kuo & Takayasu, 2023; Qiao et al., 2023) using lattice attacks. Intuitively, this is due to the fact that the attack in the NTT domain leads to an overdetermined system. While the search space for the NTT domain secret polynomial is q^n, it is significantly smaller in the normal domain due to the small coefficient distributions, as explained in Section 3.2. In particular, it is $(2 \cdot \eta + 1)^n$ for both Kyber and Dilithium, where coefficients are drawn from different distributions. We use the η to unify the notation between Kyber and Dilithium; specifically, η corresponds to η_1 in the Dilithium algorithm definition. Refer to Section 3.2 for the specific values of η.

The idea is to encode the inverse of forward NTT transformation as a lattice problem, such as SIS or LWE. In particular, Kuo & Takayasu (2023) consider the inverse NTT transformation $s = \mathbf{\Omega}_{q,n}^{-1} \hat{s}$ (see Equation (3.29)). To simplify notation, let $\mathbf{\Phi} = \Omega_{q,n}^{-1}$. Then, let $\mathbf{\Phi}_k \hat{s}_k$ and $\mathbf{\Phi}_u \hat{s}_u$ denote the known and unknown parts of this transformation, respectively:

$$s = \left[\mathbf{\Phi}_k \mid \mathbf{\Phi}_u\right] \left[\hat{s}_k \mid \hat{s}_u\right] \tag{4.1}$$

$$s = \mathbf{\Phi}_k \hat{s}_k + \mathbf{\Phi}_u \hat{s}_u \tag{4.2}$$

Table 4.1 Minimum number of known NTT domain secret coefficients $\hat{s}_{[i]}$ needed to recover whole s for Kyber768 with 1.0 success rate, using BKZ-50 (Kuo & Takayasu, 2023).

Algorithm	Υ (min. # known $\hat{s}_{[i]}$)
Kyber512	39
Kyber768	38
Kyber1024	38

where $\mathbf{\Phi}_k \in \mathbb{Z}_q^{n\times k}$, $\mathbf{\Phi}_u \in \mathbb{Z}_q^{n\times u}$ and $n = k+u$. Given the known vector $\mathbf{v} = -\mathbf{\Phi}_k \hat{s}_k \in \mathbb{Z}_q^n$ and s being short by definition, the following becomes an LWE instance,

$$\mathbf{v} = \mathbf{\Phi}_u \hat{s}_u - s \tag{4.3}$$

The LWE problem is solved by lattice reduction algorithms, such as BKZ. The number of NTT domain coefficients that is required to compute s depends on the block size used in BKZ, which also increases the run-time of the algorithm For instance, recovering $\Upsilon = 38$ NTT domain coefficients out of 128 is needed for Kyber768 when BKZ-50 is used Kuo & Takayasu (2023), as shown in Table 4.1. Here, we use Υ to denote the minimum number of correctly recovered NTT domain coefficients. Also note that BKZ-50 refers to applying the BKZ algorithm with a block size of 50.

Recall that for Kyber's incomplete NTT, the transform is applied independently to the even and odd degree coefficients. Therefore, we consider a total of 128 coefficients instead of 256, and the process described in this section is applied independently to the even and odd degree coefficients of s, as explained in Equation (3.38). Therefore, in terms of notation, we have $\mathbf{\Phi} = \Omega_{q,n/2}^{-1}$ for Kyber. Accordingly, Equation (4.4) becomes:

$$\mathbf{v}_{[:][j]} = \mathbf{\Phi}_u \hat{s}_{u[:][j]} - s_{[:][j]} \tag{4.4}$$

for $j \in \{0,1\}$.

The method in Qiao et al. (2023) differs slightly, as it leverages the forward NTT. For Dilithium3 with BKZ-50, their approach requires $\Upsilon = 79$ out of 256 secret coefficients. Readers may refer to the cited work for a detailed exploration of various trade-offs regarding the block size and run-time.

4.5. Masking Polynomial Arithmetic

In this section, we briefly explain how arithmetic masking is applied to the polynomial operations including the NTT-based polynomial multiplication, with examples from Dilithium and Kyber. Following the notation from Section 4.1, consider the polynomial multiplication $s \cdot c$.

The arithmetic shares $s^{\{0:d-1\}}$ of s are generated by sampling each coefficient uniformly at random:

$$s_{[i]}^{\{j\}} \leftarrow \mathbb{Z}_q \quad \text{for } 0 \leq i < n,\ 0 \leq j < d \tag{4.5}$$

and then computing the remaining shares as follows:

$$s_{[i]}^{\{j\}} = \left(s_{[i]} - \sum_{j=0}^{d-2} s_{[i]}^{\{j\}} \right) \pmod{q} \quad \text{for } 0 \leq i < n \tag{4.6}$$

Then, the multiplication $s \cdot c$ can be easily performed through the arithmetic shares $s^{\{0:d-1\}}$ of s. The output shares of the multiplication are computed as follows:

$$r^{\{j\}} = s^{\{j\}} \cdot c \quad \text{for } 0 \leq j < d \tag{4.7}$$

Notice that:

$$s \cdot c = \sum_{j=0}^{d-1} s^{\{j\}} \cdot c \tag{4.8}$$

This approach naturally extends to computations in the NTT domain:

$$r^{\{j\}} = \mathsf{iNTT}(\mathsf{NTT}(s^{\{j\}}) \star \mathsf{NTT}(c)) \tag{4.9}$$

We frequently use the following equality for the masked base multiplication:

$$\hat{s} \star \hat{c} = \sum_{j=0}^{d-1} \hat{s}^{\{j\}} \star \hat{c} \pmod{q} \tag{4.10}$$

Polynomial addition can be performed similarly. One of the output shares, *e.g.*,

$r^{\{0\}}$, of the result are computed as follows:

$$r^{\{0\}} = \mathsf{iNTT}(\mathsf{NTT}(s^{\{0\}}) \star \mathsf{NTT}(c)) + b \tag{4.11}$$

$$= \mathsf{iNTT}(\mathsf{NTT}(s^{\{0\}}) \star \mathsf{NTT}(c) + \mathsf{NTT}(b)) \tag{4.12}$$

where b is the polynomial added to the multiplication result. We assume that b is public and does not require masking. The other output shares are computed as in Equation (4.5). Notice that:

$$s \cdot c + b = \sum_{j=0}^{d-1} r^{\{j\}} \tag{4.13}$$

In case b is a sensitive value, it can be masked as well. In this case, the output shares are computed as follows:

$$r^{\{j\}} = \mathsf{iNTT}(\mathsf{NTT}(s^{\{j\}}) \star \mathsf{NTT}(c)) + b^{\{j\}} \qquad \text{for } 0 \leq j < d \tag{4.14}$$

or equivalently:

$$r^{\{j\}} = \mathsf{iNTT}(\mathsf{NTT}(s^{\{j\}}) \star \mathsf{NTT}(c) + \mathsf{NTT}(b^{\{j\}})) \qquad \text{for } 0 \leq j < d \tag{4.15}$$

Algorithm 10 Simplified Masked $\mathsf{Kyber.CPAPKE.Dec}(sk, c)$

Input: Secret Key $sk = (\hat{\mathbf{s}})$
Input: Ciphertext $c = (\mathbf{c}_1, c_2)$
Input: Number of shares d
Output: Plaintext shares $m^{\{0:d-1\}}$ where $m^j \in \{0,1\}^{256}$ and $m = m^{\{0\}} \oplus m^{\{1\}} \oplus \cdots \oplus m^{\{d-1\}}$

$\mathbf{u} = \mathsf{Decompress}_q(\mathbf{c}_1, d_u)$
$v = \mathsf{Decompress}_q(c_2, d_v)$
for $i = 0$ to $k-1$ **do** ▷ Masking
    $\mathbf{s}_{[i]}^{\{d-1\}} = \mathbf{s}_{[i]}$
    for $j = 0$ to $d-2$ **do**
        $\mathbf{s}_{[i]}^{\{i\}} \leftarrow \mathcal{R}_{q,n}$
        $\mathbf{s}_{[i]}^{\{d-1\}} = \mathbf{s}_{[i]}^{\{d-1\}} - \mathbf{s}_{[i]}^{\{j\}}$
    end for
end for
for $i = 0$ to $k-1$ **do** ▷ Forward NTT
    for $j = 0$ to $d-1$ **do**
        $\hat{\mathbf{s}}_{[i]}^{\{j\}} = \mathsf{NTT}(\mathbf{s}_{[i]}^{\{j\}})$
    end for
end for
for $i = 0$ to $k-1$ **do**
    $\hat{\mathbf{u}}_{[i]} = \mathsf{NTT}(\mathbf{u}_{[i]})$
end for
for $j = 0$ to $d-1$ **do** ▷ Multiplication
    $\hat{r}^{\{j\}} = 0$
    for $i = 0$ to $k-1$ **do**
        $\hat{r}^{\{j\}} = \hat{r}^{\{j\}} + \hat{\mathbf{s}}_{[i]}^{\{j\}} \star \hat{\mathbf{u}}_{[i]}$
    end for
    $r^{\{j\}} = \mathsf{iNTT}(\hat{r}^{\{j\}})$
end for
$r^{\{0\}} = v - r^{\{0\}}$
for $j = 1$ to $d-1$ **do**
    $r^{\{j\}} = -r^{\{j\}}$
end for
$m^{\{0:d-1\}} = \mathsf{MaskedCompressAndEncode}_1(r^{\{0:d-1\}}, 1)$
return $m^{\{0:d-1\}}$

A simplified version of the masked Kyber decryption algorithm (Algorithm 3) is shown in Algorithm 10. We explicitly demonstrate the backward and forward NTTs calculations through the shares as well as the accumulation during the multiplication $\mathbf{s}^T\mathbf{u}$. While masking of polynomial operations such as multiplication and additions are straightforward, some operations require special attention. These masking friendly operations are referred to as linear operations in masking literature while the others are referred to as non-linear operations. In Kyber's decryption algorithm

Algorithm 3, the Compress operation is non-linear. Many studies discuss how to mask this operation, *e.g.*, Bronchain & Cassiers (2022); Fritzmann, Van Beirendonck, Roy, Karl, Schamberger, Verbauwhede & Sigl (2022). On the other hand, Decompress and polynomial comparison other are non-linear operations from Algorithm 2 and Algorithm 6, which must be masked in CCA setting. In this thesis, we do not focus on masking of these operations, but rather on the masking of the NTT based polynomial multiplication since this is the target operation for the presented SCA attacks. In Algorithm 10, the masked computation of the compression and encoding operations are abstracted by the function MaskedCompressAndEncode.

Masking of the multiplications $c \cdot \mathbf{s_1}$ and $c \cdot \mathbf{s_2}$ in Dilithium signature generation algorithm (Algorithm 8) is similar to the Kyber decryption algorithm. Also, in Dilithium's signature generation algorithm, there are non-linear operations which are the $\mathsf{LowBits}_q$, the ExpandMask and bound checking operations.

4.6. Evaluated Implementations

We briefly summarize the real-device implementations of Kyber and Dilithium evaluated in this thesis. Consistent with the literature, our focus is on the ARM Cortex-M4 architecture, which is widely adopted in embedded devices and SCA research. We consider both unprotected and protected (masked) implementations of these algorithms. The unprotected implementations are from the well-known pqm4 library (Kannwischer et al., 2019), which provides state-of-the-art implementations of PQC algorithms for the ARM Cortex-M4 platform. The protected implementations are also based on the pqm4 library, inheriting the polynomial arithmetic routines.

In Chapter 5, we target the Kyber and Dilithium implementations from the pqm4 library, following the optimizations of Abdulrahman et al. (2022) (explained in Section 3.3.2), as well as an in-house masked version for Kyber. In Chapter 6 and Chapter 7, we target the masked Kyber implementation by Bronchain & Cassiers (2022). In Chapter 6, we also consider a modified version of this implementation, where the Plantard arithmetic proposed by Huang et al. (2022) is integrated from the pqm4 library into this masked implementation. In Chapter 7, we target the masked Dilithium implementation by Coron, Gérard, Lepoint, Trannoy & Zeitoun (2024). Further details on all target implementations are provided in the respective

chapters. Needless to say, our methods are applicable to any platform, although our experiments were conducted on the ARM Cortex-M4 architecture.

4.7. Simulated Traces

Throughout this thesis, we will be using simulated traces to evaluate the performance of presented SCA attacks. The generation of these traces is based on the principles of base multiplications in masked implementations. Algorithm 11 outlines the process for generating traces under d-share masking.

Algorithm 11 Simulated trace generation procedure for base multiplication under masking with d shares.

Input: Number of shares d, modulus q, dimension m, number of traces ν
Input: A flag f_1 indicating whether to use incomplete NTT arithmetic of not
Input: Noise mean μ and standard deviation σ, leakage function L
Input: A flag f_2 indicating whether to use central reduction or not
Output: Traces $(\mathbf{s}, \mathtt{C}, \mathtt{L})$

1: Sample secret vector $\mathbf{s} \leftarrow \mathtt{S}$, where

$$\mathtt{S} = \begin{cases} \mathbb{Z}_q^m, & f_1 \\ \mathbb{Z}_q^{m \times 2}, & \text{else} \end{cases}$$

2: $\mathtt{C} = \{\}$
3: $\mathtt{L} = \{\}$
4: **for** $0 \leq i < \nu - 1$ **do**
5: $\quad \mathbf{c} \leftarrow \mathtt{S}$ ▷ Sample public vector
6: $\quad \mathbf{s}^{\{j\}} \leftarrow \mathtt{S}$ **for** $0 \leq t < d-1$ ▷ Sample $d-1$ random secret shares
7: $\quad \mathbf{s}^{\{d-1\}} = \mathbf{s} - \sum_{t=0}^{d-2} \mathbf{s}^{\{j\}} \pmod q$
8: $\quad$ **if** f_1 **then** ▷ Complete NTT arithmetic
9: $\quad\quad$ **for** $0 \leq t < m,\ 0 \leq j < d$ **do** ▷ Compute intermediate matrix
10: $\quad\quad\quad$ **if** f_2 **then**
11: $\quad\quad\quad\quad \mathbf{x}_{[t]}^{\{j\}} = \mathbf{s}_{[t]}^{\{j\}} \cdot \mathbf{c}_{[t]} \mod^{\pm} q$
12: $\quad\quad\quad$ **else**
13: $\quad\quad\quad\quad \mathbf{x}_{[t]}^{\{j\}} = \mathbf{s}_{[t]}^{\{j\}} \cdot \mathbf{c}_{[t]} \mod^{+} q$
14: $\quad\quad\quad$ **end if**
15: $\quad\quad$ **end for**
16: $\quad$ **else** ▷ Incomplete NTT arithmetic
17: $\quad\quad$ **for** $0 \leq t < m,\ 0 \leq j < d$ **do**
18: $\quad\quad\quad$ **if** f_2 **then**
19: $\quad\quad\quad\quad \mathbf{g}_{[t]}^{\{j\}} = \mathbf{s}_{[t][0]}^{\{j\}} \cdot \mathbf{c}_{[t][1]} + \mathbf{s}_{[t][1]}^{\{j\}} \cdot \mathbf{c}_{[t][0]} \mod^{\pm} q$ ▷ as in Equation (3.36)
20: $\quad\quad\quad$ **else**
21: $\quad\quad\quad\quad \mathbf{g}_{[t]}^{\{j\}} = \mathbf{s}_{[t][0]}^{\{j\}} \cdot \mathbf{c}_{[t][1]} + \mathbf{s}_{[t][1]}^{\{j\}} \cdot \mathbf{c}_{[t][0]} \mod^{+} q$ ▷ as in Equation (3.36)
22: $\quad\quad\quad$ **end if**
23: $\quad\quad$ **end for**
24: $\quad$ **end if**
25: $\quad \mathbf{e} \sim \mathsf{N}(\mu, \sigma)$ ▷ Sample noise vector
26: $\quad$ **for** $0 \leq t < m,\ 0 \leq j < d$ **do** ▷ Compute leakage
27: $\quad\quad \mathbf{l}_{[t+tm]} = \mathsf{L}(\mathbf{g}_{[t]}^{\{j\}}) + \mathbf{e}_{[t+tm]}$
28: $\quad$ **end for**
29: $\quad \mathtt{C} = \mathtt{C} \cup \{\mathbf{c}\},\ \mathtt{L} = \mathtt{L} \cup \{\mathbf{l}\}$ ▷ Record trace
30: **end for**
31: **return** $\mathbf{s}$, $\mathtt{C}$, $\mathtt{L}$

5. FASTER ATTACKS ON INCOMPLETE NTT

This chapter is based on the publication (Tosun & Savas, 2024).

We present a novel, efficient methodology that accelerates the non-profiled power/EM side-channel attack targeting polynomial multiplication based on the incomplete NTT algorithm. Our attack applies to Kyber and certain Dilithium implementations (Abdulrahman et al., 2022). We demonstrate that the method accelerates attack run-time when compared to the existing approaches. While a conventional non-profiled side-channel attack tests a much larger hypothesis set, *i.e.* $O(q^2)$, because it needs to predict two coefficients of secret polynomials together, we propose a much faster *zero-value filtering attack* (ZV-FA), which reduces the size of the hypothesis set by targeting the coefficients individually. To achieve that, we utilize the zero-values in the publicly known vector c. We also propose an effective and efficient validation and correction technique employing the inverse NTT to estimate and modify the mispredicted coefficients. Our experimental results show that we can achieve a speed-up of **958**×over brute-force[1].

5.1. Zero-Value Filtering for Acceleration

In this section, we first show a straightforward baseline attack and then give the details of a more efficient zero-value filtering attack. Figure 5.1 presents a high-level overview of the side-channel attacks discussed in this section. The baseline scheme corresponds to the attack detailed in Section 4.3, which tests $q^2/2$ hypotheses for each pair of secret coefficients $\{\hat{s}_{[i][0]}, \hat{s}_{[i][1]}\}$ by leveraging the additive inverse trick[2].

[1]The speed-up reported in this chaper differs from that in the publication, as it is calculated against a

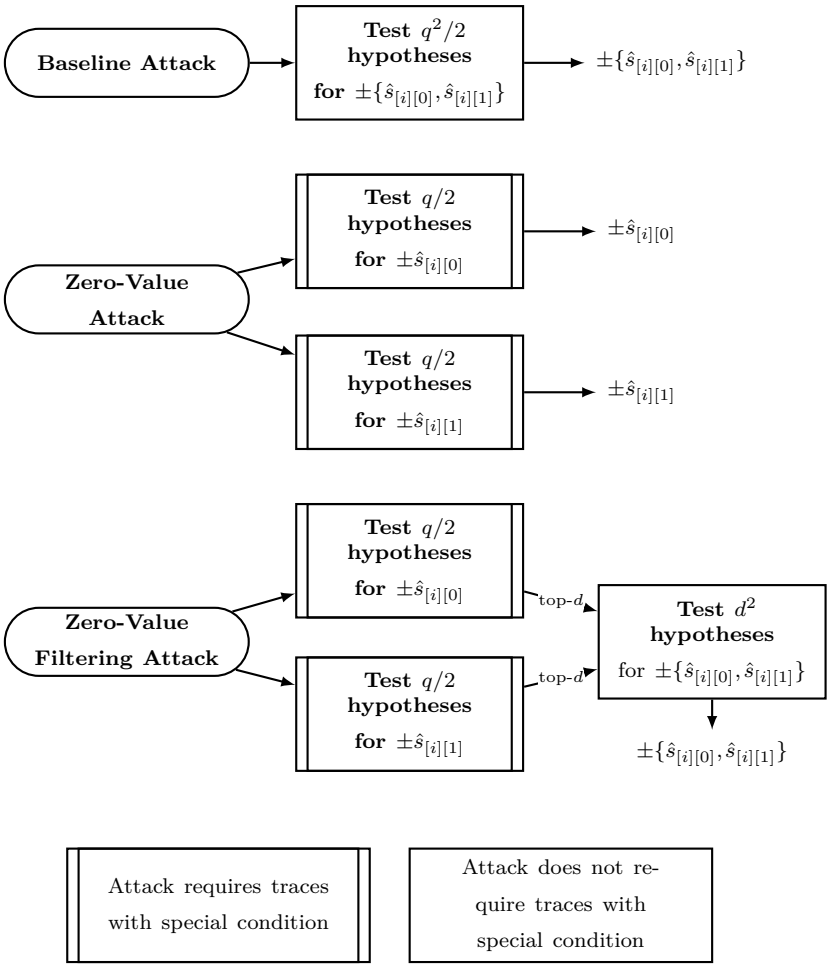


Figure 5.1 Overview of the presented attacks.

5.1.1. Decreasing the Number of Hypotheses: Zero-Value Attack

A more practical scheme compared to the baseline in terms of the attack run-time can be constructed by attacking the coefficients $\hat{s}_{[i][0]}$ and $\hat{s}_{[i][1]}$ individually, referred here as *Zero-Value (ZV) Attack*. To achieve this, we need to eliminate the effect of one of the secret coefficients from the reduction step during the NTT multiplication, whose output constitutes the chosen target function $\mathcal{G}$ (see Section 4.3). This can be accomplished by including only the traces to the attack that contain zeros in their coefficients, which multiply one of the secret coefficients. Consider Equation (3.35) to develop intuition for the proposed method. Let $\hat{r} = \hat{c} \star \hat{s}$ and assume $\hat{c}_{[i][1]} = 0 \mod q$ for some $0 \leq i < n/2$. Then $\hat{s}_{[i][1]} \cdot \hat{c}_{[i][1]} \cdot \zeta_i \mod q$ becomes 0 and $\hat{r}_{[i][0]} = \hat{s}_{[i][0]} \cdot \hat{c}_{[i][0]} \mod q$. With sufficient number of traces meeting the condition $\hat{c}_{[i][1]} = 0$

different baseline

[2] In the published work (Tosun & Savas, 2024), the baseline refers to the scheme without the additive inverse trick, whereas the variant employing it (considered the baseline in this chapter) is denoted as baseline$^+$.

All reported performance metrics have been adjusted accordingly.

mod q, predictions on $\hat{s}_{[i][0]}$ can be made independently of $\hat{s}_{[i][1]}$ for the specific value of i. The prerequisites for the ZV attack are referred to as *zero-value condition*s. Table 5.1 lists the four attacking scenarios that can be adopted. For instance, to attack $\hat{s}_{[i][0]}$, we need the condition $\hat{c}_{[i][1]} = 0 \mod q$ and use $\hat{c}_{[i][0]}$ or $\hat{c}_{[i][0]} = 0 \mod q$ and use $\hat{c}_{[i][1]}$. As the conditions are identical with attack scenarios 1 and 4, both $\hat{s}_{[i][0]}$ and $\hat{s}_{[i][1]}$ will show peaks in the results.

Table 5.1 ZV-Attack Scenarios. Probabilities are equal to $1/n$.

Attacking Scenario	Target	Condition	Used Meta	Probability
1	$\hat{s}_{[i][0]}$	$\hat{c}_{[i][1]} = 0$	$\hat{c}_{[i][0]}$	0.0039
2	$\hat{s}_{[i][0]}$	$\hat{c}_{[i][0]} = 0$	$\hat{c}_{[i][1]}$	0.0039
3	$\hat{s}_{[i][1]}$	$\hat{c}_{[i][0]} = 0$	$\zeta_i \cdot \hat{c}_{[i][1]}$	0.0039
4	$\hat{s}_{[i][1]}$	$\hat{c}_{[i][1]} = 0$	$\hat{c}_{[i][0]}$	0.0039

This approach results in a brute-force search space of size q per coefficient $\hat{s}_{[i][j]}$, which is reduced to $q/2$ by exploiting the additive inverse trick. The drawback of this method is the hardness of finding traces meeting the mentioned conditions. We mark a trace as valid for the attack if at least one coefficient in $\hat{c}$ is 0; namely,

$$\begin{aligned} &(\hat{c}_{[0][0]} = 0) \ \vee \ (\hat{c}_{[0][1]} = 0) \ \vee \ \dots \\ &\dots \ \vee \ (\hat{c}_{[n/2-1][0]} = 0) \ \vee \ (\hat{c}_{[n/2-1][1]} = 0) \quad (\text{mod } q) \end{aligned} \tag{5.1}$$

The probability for a random $\hat{c}$ to be valid depends on q and n, and can be computed by the following

$$1 - \left(\frac{q-1}{q}\right)^n \tag{5.2}$$

which leads to 0.283 for Dilithium and 0.074 for Kyber. Recall that, Dilithium's signature function applies rejection sampling, which means the target operation $c \cdot \mathbf{s_1}$ is performed several times for each signature generation, with challenge polynomials that are thrown away since the corresponding signature is rejected. However, one can retrieve the unused challenge polynomials through another side-channel attack and include them in the attack to $c \cdot \mathbf{s_1}$. We would like to note that, c is usually unprotected in the existing works from literature (Azouaoui, Bronchain, Cassiers, Hoffmann, Kuzovkova, Renes, Schneider, Schönauer, Standaert & van Vredendaal, 2023; Migliore, Gérard, Tibouchi & Fouque, 2019) and the same assumption is made by Bronchain et al. (2024). The polynomial c from rejected signatures can

be attacked using the method presented in Primas et al. (2017), which targets the calculation $\mathsf{NTT}(c)$. Moreover, since the coefficients of c are in $\{-1, 0, 1\}$, it would be relatively easier to distinguish between those. By multiplying the expected number of iterations presented in Section 3.2.2 by the probability 0.283, we find out that each signature operation contains 1.2, 1.44, 1.09 valid challenges, *i.e.* known c, on average.

On the other hand, in Kyber, the ciphertext vector of polynomials $\mathbf{u}$ is generated by the key encapsulation function (see Algorithm 2), which operates on publicly known inputs. Therefore, an attacker can brute-force the seeds used by key encapsulation to find $\mathbf{u}$ that satisfies the validity condition given in Equation 5.1.

The individual probabilities for the coefficients $\hat{c}_{[i][j]}$ being $0 \mod q$, for a valid $\hat{c}$ are another crucial factor of attack performance. Table 5.1 lists the probabilities for the aforementioned conditions, which is $1/n = 0.0039$ for any i and j. Note that, each $\hat{s}_{[i][j]}$ is attacked with the ones ensuring the corresponding zero-value conditions among the collected traces. The listed probabilities suggest that the conditions are not met very often. Intuitively, assuming that the SNR in the leakages L requires 200 traces for the attack to converge, the attacker must perform approximately 51.2K measurements on the victim's device considering the probabilities of conditions in Table 5.1. Although the attacking phase of the presented scheme is significantly faster than the baseline by a factor of q, a more optimal strategy exists, in terms of both the number of traces and the attack run-time, as presented in the next section.

5.1.2. Decreasing the Number of Traces: Zero-Value Filtering

While the ZV scheme introduced in Section 5.1.1 requires a large number of traces to retrieve the correct key exactly, alternatively having the correct key fall in top-d candidate list is relatively inexpensive in terms of the number of traces, depending on the value of d. Therefore, the ZV attack method can be used as a filtering mechanism for the following hypothesis testing, forming a two-stage attacking scheme, referred to as *Zero-Value Filtering Attack* (ZV-FA), which is formalized in Figure 5.2.

In the first stage (a.k.a. *filtering stage*), $\hat{s}_{[i][0]}$ and $\hat{s}_{[i][1]}$ are attacked individually using the ZV attack scheme as presented in the preceding section. The outcomes of these ZV attacks are denoted by $\mathtt{K}_0$ and $\mathtt{K}_1$, the sorted set of predictions for $\hat{s}_{[i][0]}$ and $\hat{s}_{[i][1]}$, respectively, based on the scores assigned by the employed distinguisher in ZV attacks. Then, in the second stage (a.k.a. *scoring stage*), a set of predictions

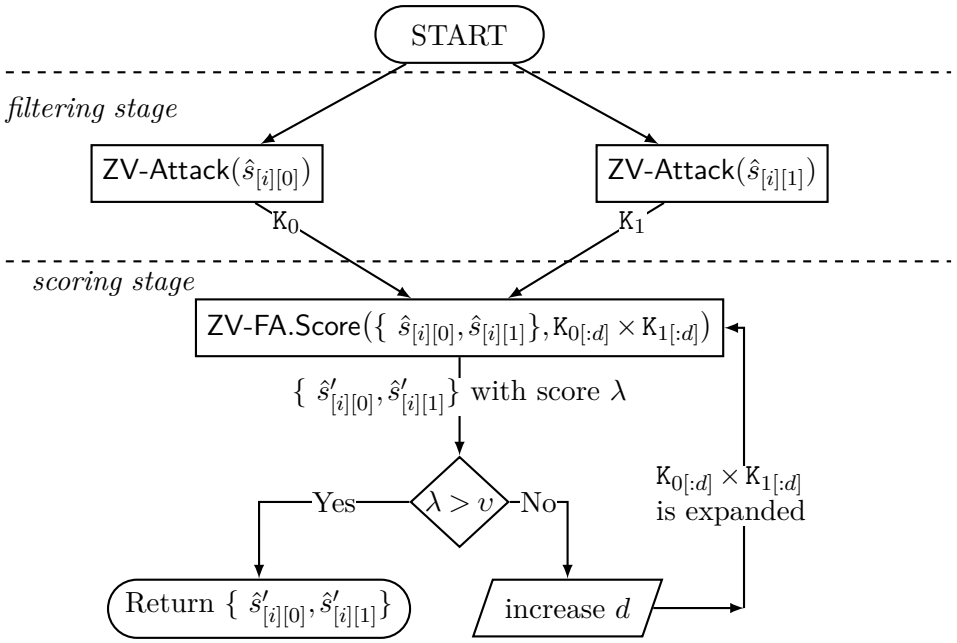


Figure 5.2 Flowchart of ZV-FA($\hat{s}_{[i]}$)

$\mathtt{K} = \mathtt{K}_{0[:d]} \times \mathtt{K}_{1[:d]}$ of size d^2 is formed for the pair $\{\hat{s}_{[i][0]}, \hat{s}_{[i][1]}\}$. Notice that $\mathtt{K}_{0[:d]}$ ($\mathtt{K}_{1[:d]}$) stands for the top scoring d predictions in $\mathtt{K}_0$ ($\mathtt{K}_1$). Afterward, $\mathtt{K}$ is scored by one of the distinguishers presented in Section 3.5.4 and Section 3.5.6. Compared to the baseline scheme, a relatively small number of hypotheses, d^2 is used for attacking $\{\hat{s}_{[i][0]}, \hat{s}_{[i][1]}\}$, as opposed to $O(q^2)$.

By the filtering stage, this method assumes that the correct key is in the top-d list of predictions of the highest scores for the ZV attack. A threshold mechanism, denoted by υ, validates the assumption through the attack output. The value of d is iteratively increased and naturally $\mathtt{K}_{0[:d]}$ and $\mathtt{K}_{1[:d]}$ get larger, until a prediction scoring greater than υ is found. By increasing d, more candidates are evaluated by the second stage, which increases the probability of having the correct key in the top-d list; naturally, the second stage takes longer to evaluate more candidates. A trivial strategy for increasing d can be doubling it. Intuitively, doubling d is acceptable for Dilithium as q is relatively small, regarding the RAM usage and response times from scoring each set of candidates. However, reducing the rate of increasing d for Kyber can be desirable, as the search space can grow quite large, considering q^2 is 23.4-bit. We explicitly state how d is updated in our experiments in Section 5.2. Needless to say, the evaluated candidates from previous trials are not included during the attack. The attack becomes identical to the baseline attack, for which the threshold is not taken into account if the scores remain below υ until d covers the whole search space.

The number of hypotheses to be tested by ZV attacks in the filtering stage is $q/2$.

Then, for each $\alpha \in \mathtt{K}_0$, we insert $-\alpha$ to $\mathtt{K}_0$, with the same rank as α. In this manner, either $\{\hat{s}_{[i][0]}, \hat{s}_{[i][1]}\}$ or its additive inverse $\{-\hat{s}_{[i][0]}, -\hat{s}_{[i][1]}\}$ is retrieved through the hypothesis testing in the scoring stage.

Compared to the ZV attack scheme, the new ZV-filtering attack is more effective with a significantly smaller number of traces. The number of traces included in the filtering stage is denoted by ν_f, while the second stage can be carried out without the zero-value conditions. As a result, it can be carried out with a sufficient number of traces to ensure that its output is reliable rather than using the entire set of valid traces, which is excessive for evaluating the score.

5.1.3. Improving ZV-FA by Using Inverse NTT to Validate Predictions

The zero-value filtering attack introduced in Section 5.1.2 relies on the assumption that a precise threshold can be found for all attacks on $\hat{s}_{[i]}$ for $0 \leq i < n/2$, which, however, may not hold in practice as a non-profiled attack is performed blindly. A possible solution to this problem is to use a conservative threshold. However, this approach is computationally expensive, and a conservative threshold can still result in false positives, albeit with a lower probability. Therefore, the attacker needs to verify the found secrets, $\mathbf{s}$ for Kyber, $\mathbf{s_1}$, $\mathbf{s_2}$ for Dilithium, by the M-LWE equation presented in Section 3.2.1 (see line 6 of Algorithm 1) and Section 3.2.2 (see line 5 of Algorithm 4), respectively. Note that this verification can only be performed after all the mentioned secrets have been attacked in all vector indices.

A more reasonable strategy for the attacker is to make use of the fact that $s = \mathsf{iNTT}(\hat{s})$ is a short polynomial. Recall that for both Dilithium and Kyber, the normal domain secret coefficients $s_{[i]}$ are in $[-\eta, \eta]$. Therefore, a small error in the prediction will diffuse through the inverse NTT computation and ruin the coefficients of the output polynomial, empowering the attacker to efficiently validate the attack output.

Figure 5.3 illustrates the flowchart of the ZV-FA from a higher-level perspective with validation using the inverse NTT. Let $\hat{s}'$ denote a prediction to $\hat{s}$, formed after completing the individual ZV attacks to $\hat{s}_{[i][0]}$ and $\hat{s}_{[i][1]}$ for all i at Step 1. To validate $\hat{s}'$, $\mathsf{iNTT}(\hat{s}')$ is computed, and the shortness property is sought in the resulting polynomial. If the found polynomial is not validated the mispredicted pair of coefficients in $\hat{s}'$ is approximated and re-attacked to correct it. The approximation is performed by selecting the pair of coefficients with the minimum score. The

current score for the prediction $\hat{s}'_{[i]}$ is denoted by λ_i. Observe that the outputs of ZV attacks are immediately scored at Step 2 if the shortness check fails. This initial scoring step is needed to be able to compare the ZV scores with ZV-FA scores, which are originally in distinct scales. The same distinguisher with ZV-FA.Score is applied to $\hat{s}'_{[i]}$ using the top scorer candidates from Step 1, $\mathtt{K}_{i,0[0]}$ and $\mathtt{K}_{i,1[0]}$, with the same number of traces to get a comparable score with ZV-FA. In this manner, the attack terminates without re-attacking predictions which are already correctly predicted by ZV. Note that $\mathtt{K}_{i,0}$ ($\mathtt{K}_{i,1}$) is the set of predictions for $\hat{s}_{[i][0]}$ ($\hat{s}_{[i][1]}$) sorted for the ZV attack scores, slightly modifying our notation from the previous section, as the coefficient index i is added to superscript.

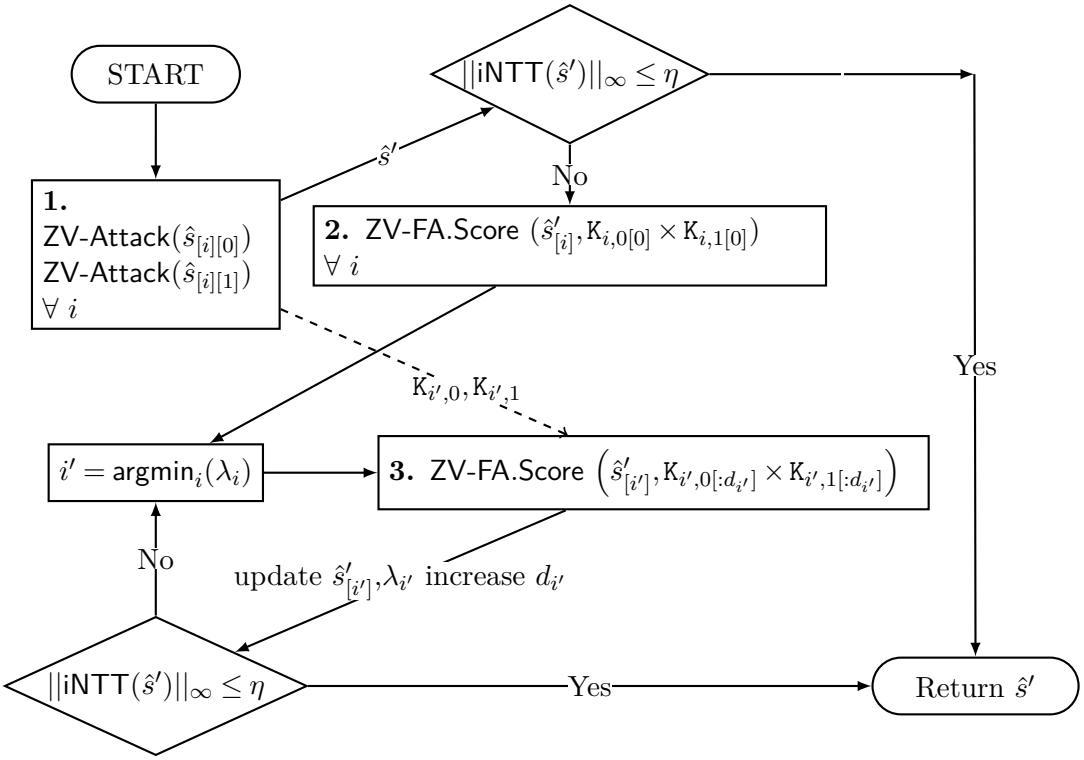


Figure 5.3 ZV-FA for the whole vector of polynomials $\hat{s}$ with the application of inverse NTT validation

The index of the minimum scoring pair from $\hat{s}'$ is found by computing $i' = \mathsf{argmin}_i(\lambda_i)$ and ZV-FA.Score is performed on $\hat{s}_{[i']}$ at Step 3 to replace $\hat{s}'_{[i']}$. Here, we skip the filtering stage of ZV-FA (see Figure 5.2) because it is indeed performed at Step 1 and therefore $\mathtt{K}_{i',0}$ and $\mathtt{K}_{i',1}$ are known. $\hat{s}'_{[i']}$ is updated if a better scoring prediction is found at the current invocation of ZV-FA.Score. At the same time, $d_{i'}$ is updated as in Figure 5.2 (Again, the coefficient index i is included in the notation to differentiate between d for $\hat{s}_{[i]}$). Notice that the ZV-FA.Score can be performed for the same $\hat{s}_{[i]}$ multiple times. However, the scoring is performed with distinct d_i at each invocation of ZV-FA.Score to expand the search for the actual secret. Recall that from the previous section, predictions for $\hat{s}'_{[i]}$ that are evaluated previously are

not included in the attack. The prediction $\hat{s}'_{[i]}$ is guaranteed to be corrected by subsequent applications of ZV-FA.Score if the correct value for $\hat{s}'_{[i]}$, is the top-scorer among $q \cdot q/2$ candidates.

The scheme is equivalent to ZV attack in terms of run-time if the found polynomial is validated after Step 1. On the other hand, in the worst case, the scheme tests $q \cdot (q/2) \cdot (n/2)$ hypotheses in total for all i, via iterations of Step 3, thus it's becoming equivalent to the baseline. The differentiation between both directions depends on the number of filtering traces, ν_f. Note that the threshold v presented in the previous section is not needed in the improved scheme thanks to inverse NTT validation. The application of inverse NTT as a reliable and efficient method of verification renders the ZV-FA fault-tolerant. This method ensures the preservation of accuracy regardless of the choice of ν_f, enhancing the attack performance.

5.2. Results

In this section, we present the results obtained after implementing the above-mentioned attacks in a realistic experimental setting, on Dilithium3 and Kyber768. Moreover, we apply our attack on a masked version of Kyber768.

5.2.1. Target Implementation

We targeted the widely used pqm4 library (Kannwischer et al., 2019)[3], which offers state-of-the-art implementations of post-quantum cryptographic (PQC) algorithms for the ARM Cortex-M4 platform. The specific version of the library analyzed incorporates the optimizations described by Abdulrahman et al. (2022). Notably, this implementation employs incomplete NTT arithmetic for the operations $c \cdot \mathbf{s_1}$ and $c \cdot \mathbf{s_2}$ in Dilithium, as detailed in Section 3.3.2. Both Kyber and Dilithium implementations are extensively optimized using Cortex-M4 specific instructions, which are further discussed in Section 3.3.2.

[3] `https://github.com/mupq/pqm4`, commit hash: **3bfbbfd**

5.2.2. Experimental Setup

We employed Analog Devices' MAX32520[4] as the victim device to run pqm4's Dilithium signature and Kyber key decapsulation implementations. The first-order masked implementation of Kyber is developed on top of the pqm4's implementation by randomly splitting s. For EM trace collection, LeCroy WavePro HD oscilloscope[5] and Langer ICR HH500-6[6] nearfield micro-probe were used. Sampling rate of the oscilloscope was set to 10 GS/s and 1 GS/s, yielding 83.33 and 8.33 samples per clock, for unprotected and protected implementations, respectively. We set up a trigger at the start of the base multiplication in the target implementations of both Dilithium and Kyber to record the timing samples relevant to our attack. The scared library[7] is used for analysis and attack, running on a computer equipped with 64 GB RAM and AMD Ryzen 9 5900X 12-Core Processor clocked at 3.70 GHz.

5.2.3. Pre-processing and Analysis

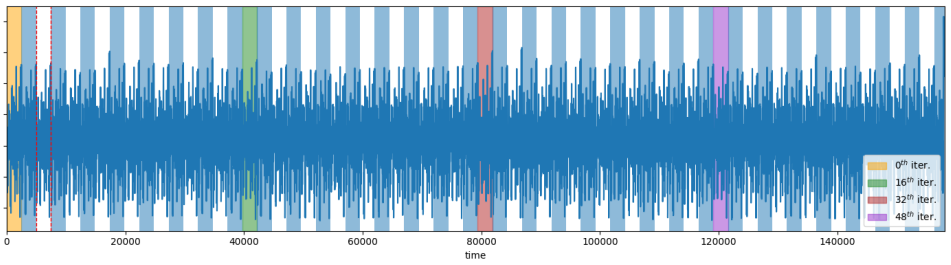
To cope with the adverse effects of misalignment over time, we performed the following pre-processing steps for both algorithms: 1) pattern detection, 2) signal filtering, and 3) extraction around peaks. A reference pattern is set by band-pass filtering the first trace between 100 MHz and 140 MHz and applying moving variance to it. The traces are aligned based on the reference pattern. Then, 64 peaks (128 for first-order protected Kyber), which correspond to iterations of the base multiplication (Equation (3.35), Equation (3.36)) (loop is unrolled by a factor of two), are detected and sequential points after each peak are combined. Figure 5.4 highlights the iterations over the average of pre-processed traces for Dilithium, conforming with the pre-knowledge on the implementation. On the other hand, iterations of the base multiplication for (masked) Kyber, are observed (for both shares) in Figure 5.5. Given the clear visibility of the iterations of the base multiplication over time samples, it is possible to conduct individual attacks on $\hat{s}_{[i]}$ in time regions associated with each iteration. Note that, partitioning the attack range over time is critical for

[4] https://www.analog.com/en/products/max32520.html

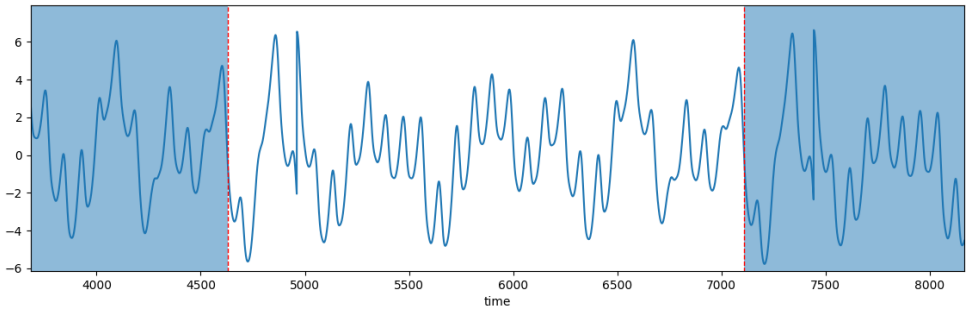
[5] https://teledynelecroy.com/oscilloscope/wavepro-hd-oscilloscope

[6] https://www.langer-emv.de/en/product/near-field-microprobes-icr-hh-h-field/26/icr-hh500-6-near-field-microprobe-2-mhz-to-6-ghz/108

[7] https://pypi.org/project/scared/



(a) Full EM trace.



(b) Zoomed area.

Figure 5.4 The mean EM trace associated with the execution of the base multiplication function of the attacked incomplete NTT-based implementation of Dilithium, `_asymmetric_mul_16_loop`. Iterations of the function are highlighted.

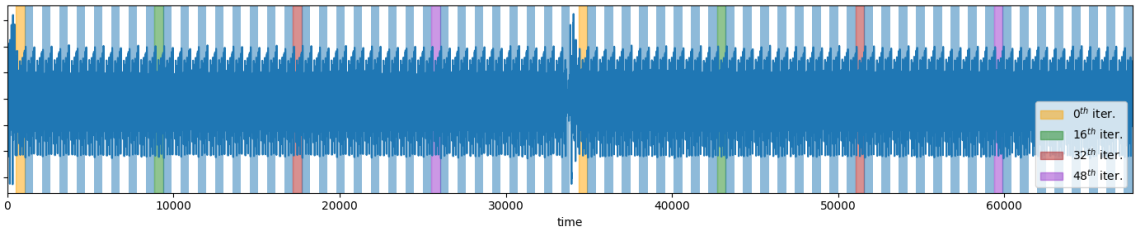
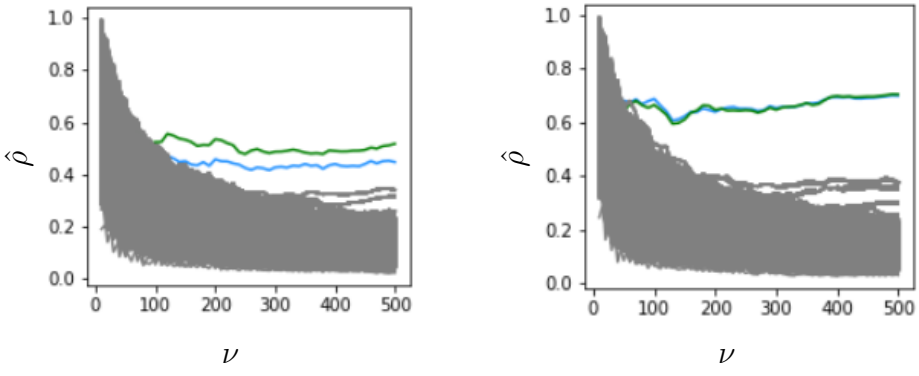


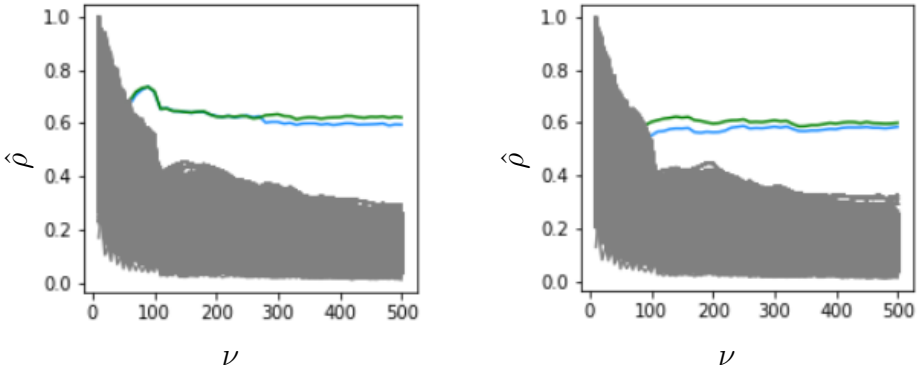
Figure 5.5 The mean EM trace associated with the execution of the base multiplication function of the attacked implementation of Kyber, `frombytes_mul_asm_acc_32_16`. Iterations of the function are highlighted for both shares.

the presented performance results of all schemes.

Upon observation, we use the concatenation of $\hat{r}_{[i][0]}$ and $\hat{r}_{[i][1]}$ (see Section 4.3) as the PoI for attacking Dilithium, while we use $\hat{r}_{[i][0]}$ for Kyber. As an initial analysis, we performed the baseline$_{\text{CPA}}$ on $\hat{s}_{[0]}$ and $\hat{s}_{[1]}$. Note that we use the subscript to denote the distinguisher used for the baseline attack. The convergence patterns of the retrieved secrets are presented in Figure 5.6.



(a) Dilithium, $\hat{s}_{[0]}$ (b) Dilithium, $\hat{s}_{[1]}$



(c) Kyber, $\hat{s}_{[0]}$ (d) Kyber, $\hat{s}_{[1]}$

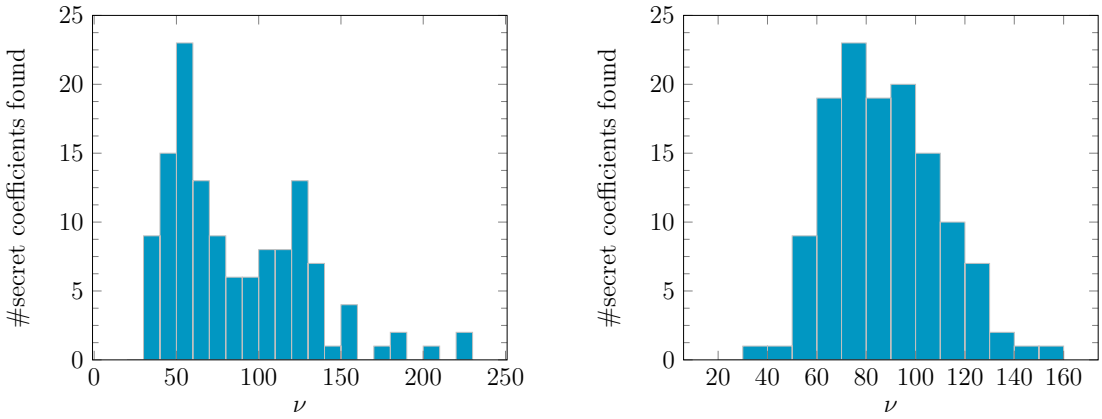
Figure 5.6 Key convergence of $\text{baseline}_{\text{CPA}}$ on Dilithium and Kyber for $\hat{s}_{[0]}$ and $\hat{s}_{[1]}$. Green lines denote the correct hypothesis while the blue line denotes its additive inverse.

5.2.4. Attack and Performance

In this section, we present the performance of the proposed attacks. We start with the first-order attacks on unprotected implementations of Kyber and Dilithium, followed by the second-order attacks on the masked implementation of Kyber.

5.2.4.1. First-order

For the first-order attacks that target unprotected implementations of Dilithium and Kyber, we use CPA as the distinguisher. We start the evaluation by the performance of the $\text{baseline}_{\text{CPA}}$ scheme. Figure 5.7 illustrates the distribution of the required number of traces $\hat{s}_{[i]}$ needed to converge in our experiments. Note that, the histograms are computed based on a single $\hat{s}$ and varying i. Accordingly, 220 and 150 traces are needed the retrieve whole $\hat{s}$ for Dilithium and Kyber, respec-



(a) Dilithium (b) Kyber

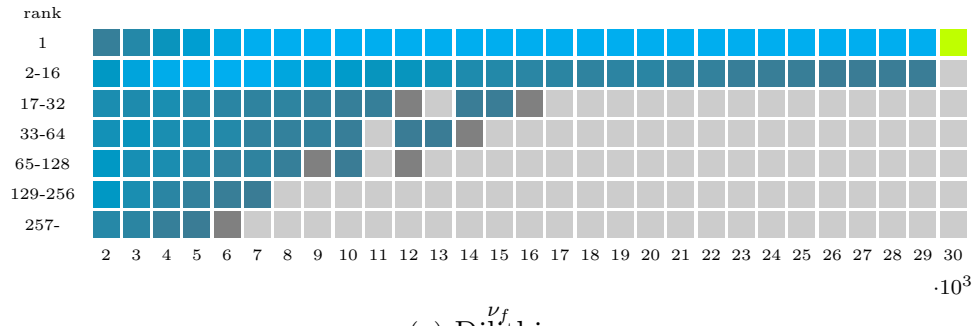
Figure 5.7 Histograms of the required number of traces ν for the baseline$_{\text{CPA}}$ on Kyber and Dilithium to to succeed.

tively. Notice that, these numbers are the maximums of the values presented in the histograms. However, a significant portion of the secret coefficients, $\hat{s}_{[i]}$, are indeed revealed with fewer traces, around 50. The performance of the baseline$_{\text{CPA}}$ scheme is reported in Table 5.2. In terms of accuracy, it exhibits flawless performance and do not pose any concerns regarding accuracy. However, in terms of run-time, the performance of the attack is moderate. Nevertheless, retrieving $\mathbf{s_1}$ and $\mathbf{s_2}$ requires approximately 4.37 hours for Dilithium while it takes 22.4 hours to attack $\mathbf{s}$ for the Kyber case. Note that the baseline$_{\text{CPA}}$ is equivalent to the attack presented by Mujdei et al. (2024) against the same implementation of Kyber. Our application of the conventional approach is slightly better than that work both in terms of attack run-time and number of traces.

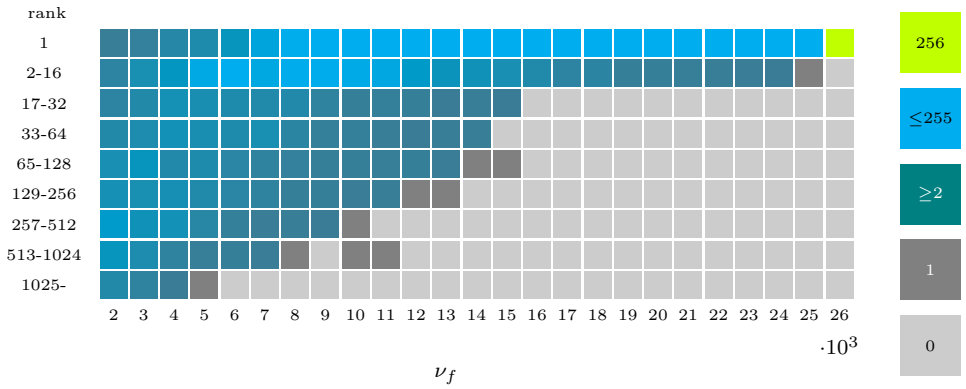
For the application of ZV-FA to Dilithium (to Kyber), the attack scenarios 1 and 2 (scenario 1 for Kyber) from Table 5.1 are employed for attacking the lower degree coefficients of the polynomials, specifically $\hat{s}_{[i][0]}$ for any $0 \leq i < 128$, while the scenarios 3 and 4 (scenario 3) are utilized for the higher degree coefficients $\hat{s}_{[i][1]}$. It should be noted that scenarios 1 and 4 are not independently executed, as they represent the same attack. The outcomes of different scenarios are combined by

Table 5.2 Run-time performance of the baseline$_{\text{CPA}}$ attack on unprotected implementations of Kyber768 and Dilithium3.

Algorithm	Method	ν	Runtime($\hat{s}_{[i]}$)	Runtime($\hat{s}$)	
					Runtime($\mathbf{s_1}$,$\mathbf{s_2}$)
Dilithium3	This Work	220	11.18 s	24 m	≈4.37 h
					Runtime($\mathbf{s}$)
Kyber768	This Work	150	3.5 m	7.45 h	≈22.4 h
Kyber	(Mujdei et al., 2024)	200	5 m	10.7 h	



(a) Dilithium



(b) Kyber

Figure 5.8 Histograms of the ranks of correct hypotheses for $\hat{s}_{[i][j]}$ in $\mathtt{K}_{i,j}$ during filtering stage of ZV-FA on Kyber and Dilithium, *i.e.* the correct secret among results of ZV attacks, w.r.t. ν_f.

multiplying their respective results. For both schemes, we use $\nu = 500$ traces for $\mathsf{ZV-FA.Score}$, and d_i is doubled to increase it after each call to $\mathsf{ZV-FA.Score}$ (see Figure 5.3). Note that, we use slightly more traces compared to the convergence of the baselines (see Figure 5.6). This is needed to have the score of $\hat{s}_{[i]}$ converged and become comparable with the scores of predictions to other secret coefficients. Observe from Figure 5.6 that the SNR of even and odd coefficients are different in Dilithium. Therefore, we scale the scores onto the same range by multiplying the odd coefficients by $\lambda_0/\lambda_1 \approx 5/7$.

Thanks to the inverse NTT validation and correction mechanism presented in Section 5.1.3, The ZV-FA's accuracy is independent of the number of traces used for filtering ν_f, which, however, determines its performance. Recall that, the performance of the ZV-FA scheme strongly depends on the effectiveness of the filtering stage. Figure 5.8 shows that, even with moderate values of ν_f, the correct value for the attacked secret pair is discovered within the top-d for a significant portion of the secret coefficients and practical values of d in terms of performance. Particularly for Dilithium and $\nu_f = 5\text{K}$, 203 of 256 (80%) secret coefficients are retrieved

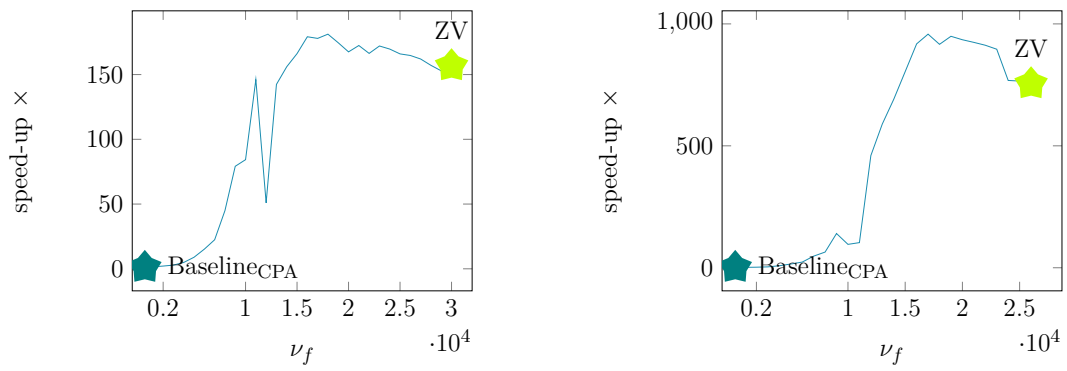
in top-64, which corresponds to $(769-64)/769$ (92%) reduction in the search space from the baseline to $\mathsf{ZV-FA.Score}$. Similarly, for Kyber and the same value of ν_f, 202 coefficients are in the top-256, leading to (92%) reduction in the search space.

Experimental results indicate that the ZV filtering can substantially decrease the attack response time up to three orders of magnitude, depending on the number of filtering traces ν_f available in the system. The trade-off between ν_f and speed-up is illustrated in Figure 5.9 for both attacked algorithms. Observe that the trade-off suggests the same pattern for Dilithium and Kyber. The ZV-FA approaches to the ZV attack as ν_f increases and approaches to the baseline as ν_f decreases. Notably, even a small number of traces can significantly improve baseline performance. For example with $\nu_f = 5K$ the collection of which is feasible, the ZV-FA provides a speed-up of $\mathbf{9}\times$ and $\mathbf{15}\times$ for Dilithium and Kyber, respectively.

When more valid traces are available in the system, particularly with $\nu_f = 18K$, ZV-FA achieves a speed-up of $\mathbf{181}\times$ for Dilithium over the $\text{baseline}_{\text{CPA}}$. We underline that, the achieved speed-up can save approximately 261 minutes (≈ 4.35 hours) considering the retrieval of whole $\mathbf{s_1}$ and $\mathbf{s_2}$. Recall that $k = 6$ and $l = 5$ for Dilithium3, which means $\mathbf{s_1}$ and $\mathbf{s_2}$ consist of 5 and 6 secret polynomials, respectively. As for Kyber (recall that $k = 3$ for Kyber768, which means $\hat{s}$ has 3 elements.), our scheme is more favorable as q is roughly 2-bit larger compared to Dilithium3. With $\nu_f = 17K$, ZV-FA achieves $\mathbf{958}\times$ speed-up over the $\text{baseline}_{\text{CPA}}$. It saves roughly 22.38 hours of computation time, considering all the elements of Kyber's secret vector of polynomials $\hat{s}$.

On the other hand, ZV-FA reduces the number of traces needed for the ZV attack while the run-time performance is slightly improved. Observe that the ZV attack is successful with $\nu_f = 30K$ for Dilithium which brings up a speed-up of $\mathbf{157}\times$. The same speed-up is achieved by ZV-FA with $\nu_f \approx 14K$. Similarly for Kyber, the ZV attack is successful with $\nu_f = 26K$ accelerating the $\text{baseline}_{\text{CPA}}$ by $\mathbf{754}\times$ while ZV-FA reaches the same speed-up with $\nu_f \approx 14.5K$.

Recall also that another study by Chen et al. (2021) targets Dilithium with a non-profiled attack. However, the target implementation uses the original 23-bit coefficient modulus of Dilithium. Therefore, our study is not comparable to Chen et al. (2021) as the target implementations are different. On the other hand, while their method accelerates the baseline approach about 16 times, ours provides a speedup of more than two orders of magnitude. Consequently, we expect their method would not be as effective as ours when an incomplete NTT method is used in the implementation.



(a) Dilithium (b) Kyber

Figure 5.9 Speed-up of ZV-FA on Kyber and Dilithium w.r.t. ν_f. The performance of baseline$_{\text{CPA}}$ and ZV attack are marked.

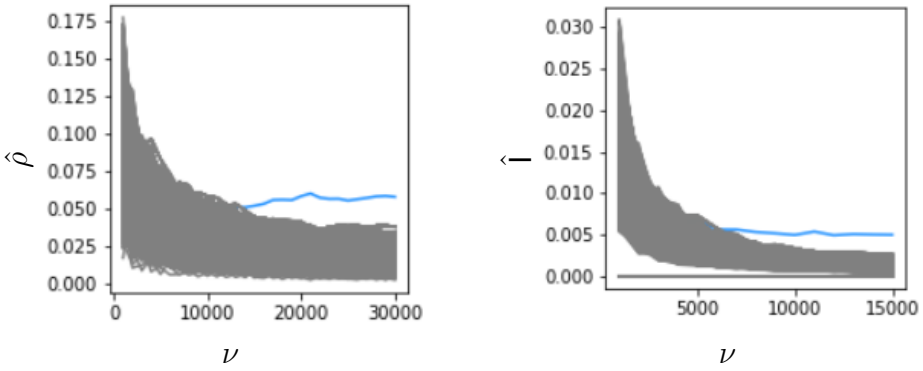
Table 5.3 Run-time performance of baseline attacks with MMIA and HOCPA on first-order masked Kyber768

Method	i	ν	Runtime($\hat{s}_{[i]}$)	Runtime($\hat{s}_{[i]}$)$\cdot n/2 \cdot k$ ≈Runtime($\hat{s}$)
Baseline$_{\text{HOCPA}}$	0	14K	78 m	≈21 d
Baseline$_{\text{HOCPA}}$	1	23K	129 m	≈34 d
Baseline$_{\text{MMIA}}$	0	7K	182 m	≈48.5 d

5.2.4.2. Second-order

To discuss the efficiency of our attack in the protected case, we first present the performance of the baseline schemes thereof in Table 5.3. Due to long running times, we perform the evaluation only for $\hat{s}_{[0]}$ and $\hat{s}_{[1]}$. The last column approximates the required amount of time to break whole secret key $\hat{s}$, based on the statistics of the attacks to $\hat{s}_{[0]}$ and $\hat{s}_{[1]}$. For instance, retrieving $\hat{s}$ would roughly take 68 days if all coefficients were retrieved by 23K traces as $\hat{s}_{[1]}$. Therefore, the baseline scheme stands as an impractical option. The speed-up of ZV-FA is computed based on the average number of traces needed by baseline$_{\text{HOCPA}}$ over the analyzed coefficients, $(23+14)/2 = 18.5$K.

Observe from Table 5.3 and Figure 5.10 that MMIA is superior to HOCPA in terms of the number of traces while HOCPA runs faster. Therefore, we use MMIA as the distinguisher in the filtering stage, differently from the unprotected case. We utilize MMIA with two bins and select the bins such that it partitions the samples into equal parts, *i.e.* halves. As the iterations of the base multiplication are easily distinguishable through the mean trace as depicted by Figure 5.5, we use constant offset for combining points over time samples. We would like to note that we observe that the MMIA is quite efficient in this setting. However, we leave a comprehensive



(a) $\text{baseline}_{\text{HOCPA}}$ (b) $\text{baseline}_{\text{MMIA}}$

Figure 5.10 Key convergence of $\text{baseline}_{\text{HOCPA}}$ and $\text{baseline}_{\text{MMIA}}$ on masked Kyber for $\hat{s}_{[0]}$. Blue line denotes the additive inverse of the correct hypothesis.

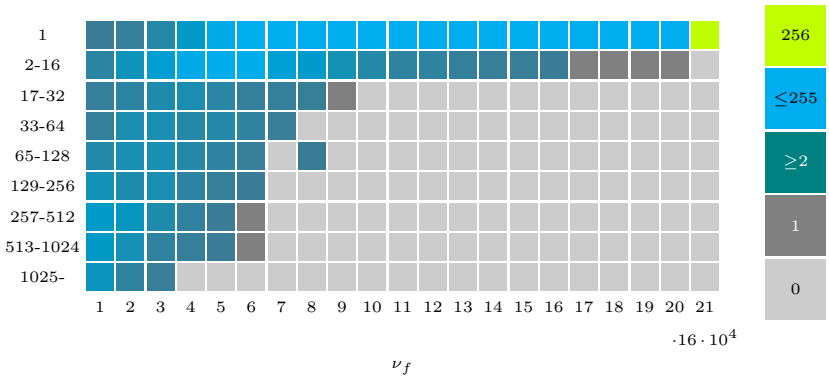


Figure 5.11 Histograms of the ranks of the correct hypotheses for $\hat{s}_{[i][j]}$ in $\mathtt{K}_{i,j}$ during filtering stage of ZV-FA on masked Kyber, with respect to ν_f.

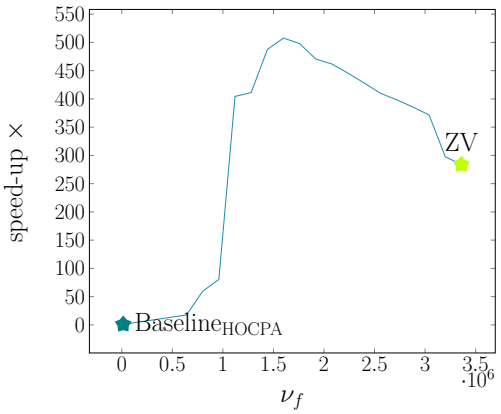


Figure 5.12 Speed-up of ZV-FA on Masked Kyber w.r.t. ν_f.

study on MMIA in comparison with HOCPA regarding arithmetic masking as a future work. On the other hand, we use HOCPA in the scoring stage, considering its superior run-time performance, and we employ $\nu = 50K$ traces for scoring. Recall that, significantly more traces are available during scoring as zero-value conditions are not sought therein.

Figure 5.11 demonstrates the distribution of the ranking of the correct hypothesis for $\hat{s}_{[i][j]}$, among the sorted results from the filtering stage of ZV-FA, $\mathtt{K}_{i,j}$. We observe that the filtering stage is quite effective as in the unprotected case. Particularly with $\nu_f = 480K$, for 213 out of 256 secret coefficients, the correct hypothesis is in top-256, *i.e.* $\mathtt{K}_{i,j[:256]}$, leading to 92% reduction in search space. Figure 5.12 depicts the speed-up values achieved over the baseline$_{\text{HOCPA}}$. We observe that ZV-FA reaches **508**$\times$ speed-up over baseline$_{\text{HOCPA}}$, with $\nu_f = 1.6M$. If those many traces are not available to the attacker, ZV-FA is still more practical compared to the baseline$_{\text{HOCPA}}$. For instance, a speed-up of **18**$\times$ is observed with a relatively less number of filtering traces, $\nu_f = 640K$.

5.2.4.3. Notes on HOCPA and MMIA

We note that the HOCPA applied in this chapter is not efficient in terms of the required number of traces. Specifically, we employ the mean-free product combination described in Section 3.5.6.1, but use HW_β as the prediction function instead of the optimal prediction function $\mathsf{F}^2_{\text{opt}}$. Nevertheless, this choice does not affect the relative performance of ZV-FA compared to the baseline, since both approaches rely on the same prediction function. A study of the optimal prediction functions $\mathsf{F}^d_{\text{opt}}$ has been conducted in Chapter 6 for $d = 2$ and in Chapter 7 for $d > 2$. Employing these optimal functions would further improve the performance of ZV-FA.

For MMIA, on the other hand, we used the prediction function HW_β, which is uncommon given that MMIA is a generic distinguisher. Conducting these experiments without a prediction function—while employing a clever histogram binning strategy (potentially leveraging the results presented in Chapter 6) could further improve the accuracy of MMIA.

6. EXPLOITING THE CENTRAL REDUCTION

This chapter is based on the publication (Tosun et al., 2024).

We question the side-channel security of **central reduction** technique, which is widely adapted in efficient implementations of LBC (see Section 3.4). We show that the central reduction leads to a vulnerability by creating a strong dependency between the power consumption and the sign of sensitive intermediate values. We exploit this dependency by introducing the novel **absolute value** prediction function, which can be employed in higher-order non-profiled multi-query SCA attacks. Our results reveal that – compared to classical reduction algorithms – employing the central reduction scheme leads to a two-orders-of-magnitude decrease in the number of required SCA measurements to exploit secrets of masked implementations. We particularly show that our approach is valid for the prime moduli employed by Kyber and Dilithium, the lattice-based post-quantum algorithms selected by NIST. We practically evaluate our introduced approach by performing second-order non-profiled attacks against an open-source masked implementation of Kyber on an ARM Cortex-M4 micro-processor. In our experiments, we revealed the full secret key of the aforementioned masked implementation with only **250** power traces without any forms of profiling or choosing the ciphertexts.

6.1. Leakage of Signed Integers Modulo q

In this section, we study the distribution of HW of the signed representation of integers modulo q, *i.e.* the effect of central reduction on HW. Accordingly, we compare signed and unsigned arithmetic in terms of SCA leakage. Throughout this chapter, we assume that device leakage is a function of HW, $\mathsf{L} = \mathsf{HW}$ in Equation (3.45):

$$L = \mathsf{HW}_{\beta}(X) + \mathsf{N}(\mu, \sigma) \tag{6.1}$$

where β denotes the machine word size.

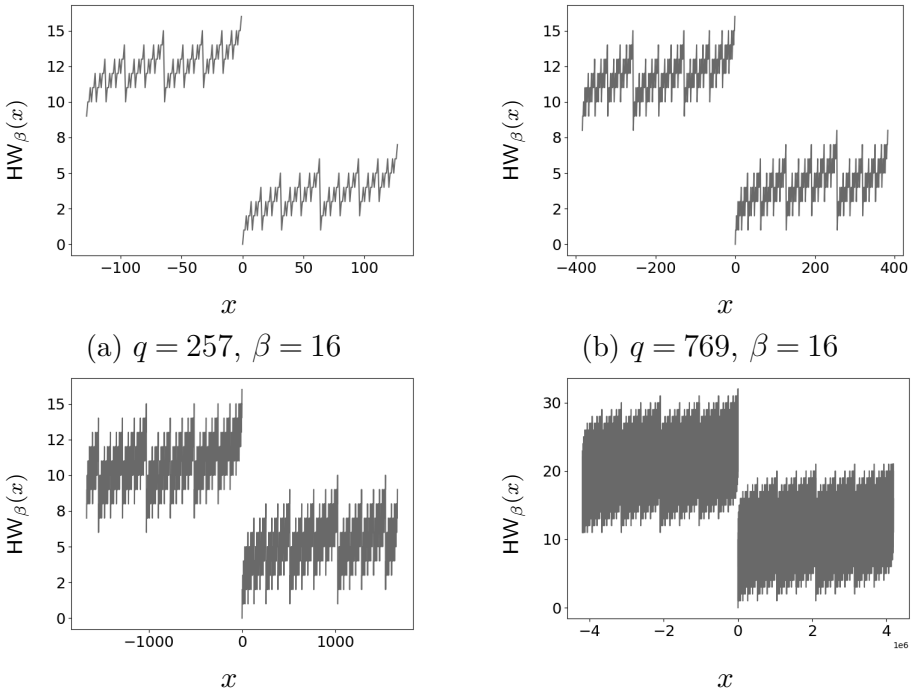
We evaluate the primes that are employed in Kyber and Dilithium with $q = 3329$ and $q = 8380417$, respectively, and study the carrier primes that are employed for Dilithium to perform short polynomial arithmetic (Abdulrahman et al., 2022), namely $q = 257$ for Dilithium2 and Dilithium5, and $q = 769$ for Dilithium3. We should note that masking the short polynomials in their range is possible (Aikata, Basso, Cassiers, Mert & Roy, 2023). One important factor for computing the HW of negative integers is the machine word size β which does not have any effect on the HW of positive integers. The machine word size is usually $\beta = 16$ for $q = 257$, $q = 769$, and $q = 3329$ while $\beta = 32$ for $q = 8380417$ in software implementations. Unless otherwise stated, we take these values for β in the studied adversary model. However, we discuss the role of distinct values of β in the SCA leakages.

6.1.1. HW as a Sign Indicator

The main observation that led to this study is the clear separation of HW of the non-negative side of ${}^{\pm}\mathbb{Z}_q$, namely $[0, q/2]$ and the negative side $[-q/2, 0)$. As an intuition, consider $q = 257$; the positive interval of ${}^{\pm}\mathbb{Z}_{257}$ corresponds to $[0, 128]$, for which the maximum HW is $\mathsf{HW}_\beta(127) = 7$. In other words, the HW of integers $[0, 128]$ lies in $[0, 7]$. On the other hand, the negative side of ${}^{\pm}\mathbb{Z}_{257}$ corresponds to $[-128, 0)$, where the HW are in the range of $[9, 16]$ assuming 2's complement representation with machine word size $\beta = 16$. Consequently, the HW of a number in ${}^{\pm}\mathbb{Z}_{257}$, reveals its sign immediately. Figure 6.1 visualizes our observation for all the primes analyzed in this study. Note that there is an overlap between the HW ranges $[-q/2, 0)$ and $[0, q/2]$, for $q = 769$, $q = 3329$, and $q = 8380417$. For instance, the HW of non-negative integers in ${}^{\pm}\mathbb{Z}_{3329}$ are distributed in $[0, 10]$ while that of negative integers are in the range $[6, 16]$. Therefore, there is an overlap for five possible HWs in the interval $[6, 10]$ out of a total of 17 possible values assuming the machine word size $\beta = 16$.

Based on our observation in Figure 6.1, we write the following equality for $\mathsf{HW}_\beta(x)$ where $x \in {}^{\pm}\mathbb{Z}_q$.

$$\mathsf{HW}_\beta(x) = \mathsf{S}(x) \cdot \gamma + (1 - \mathsf{S}(x)) \cdot (\beta - \gamma) + e, \tag{6.2}$$



(a) $q = 257$, $\beta = 16$ (b) $q = 769$, $\beta = 16$

(c) $q = 3329$, $\beta = 16$ (d) $q = 8380417$, $\beta = 32$

Figure 6.1 Distribution of HW of integers in $[-q/2, q/2]$ in 2's complement representation for different moduli q.

for some inner-cluster error term $e \in \mathtt{E}$ where $\mathsf{E}[e] = 0$ and $0 < \gamma < \beta/2$. S is defined to be the sign function:

$$\mathsf{S}(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{6.3}$$

γ stands for the mean of HWs given a uniformly random $X \in {}^{\pm}\mathbb{Z}_q$ is non-negative, *i.e.* $\gamma = \mathsf{E}[\mathsf{HW}_\beta(X) \mid \mathsf{S}(X) = 1]$. Similarly, let γ^- denote the mean HW given X is negative, $\gamma^- = \mathsf{E}[\mathsf{HW}_\beta(X) \mid \mathsf{S}(X) = 0]$. Table 6.1 demonstrates the values of γ and γ^- depending on q. Note that $\gamma^- \approx \beta - \gamma$, which allows us to simplify Equation (6.2). Naturally, γ^- approaches γ as the so-called gap $\mathsf{M}(q, \beta) = \beta - \log(q)$ decreases. On the other hand, the expected value $\mathsf{E}[|e|]$ is not affected by β while it slightly increases as q gets larger. Additionally, whether the integers modulo q are represented in signed or unsigned form has no impact on this expected value.

We would like to note that the argument made in this section does not apply to the unsigned representation of integers modulo q, namely $\mathbb{Z}_q$. It can be seen in Figure 6.2 that no clear ranges can be identified for unsigned integers when observing their HW. In LBC, the upper half of $\mathbb{Z}_q$ is considered as negative, namely $[-q/2 + q, q - 1]$. However, Table 6.1 shows that the mean of HW of negative side, denoted by γ^*, is

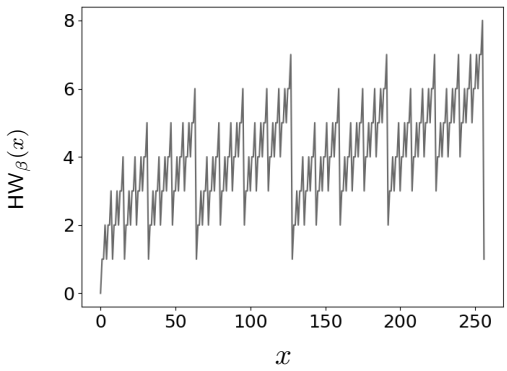


Figure 6.2 Distribution of HW of integers in $[0, q)$ for $q = 257$.

Table 6.1 The mean and standard deviation of HW for positive and negative integers with different q. γ denotes the mean of HWs for positive integers in ${}^{\pm}\mathbb{Z}_q$, namely $[0, q/2]$. γ^- denotes the mean of HWs for negative integers in ${}^{\pm}\mathbb{Z}_q$, namely $[-q/2, -1]$. γ^* denotes the mean of HWs for $[q/2+1, q-1]$, the set of integers in $\mathbb{Z}_q$ which are considered to be negative in LBC, and e is the error term defined in Equation (6.2).

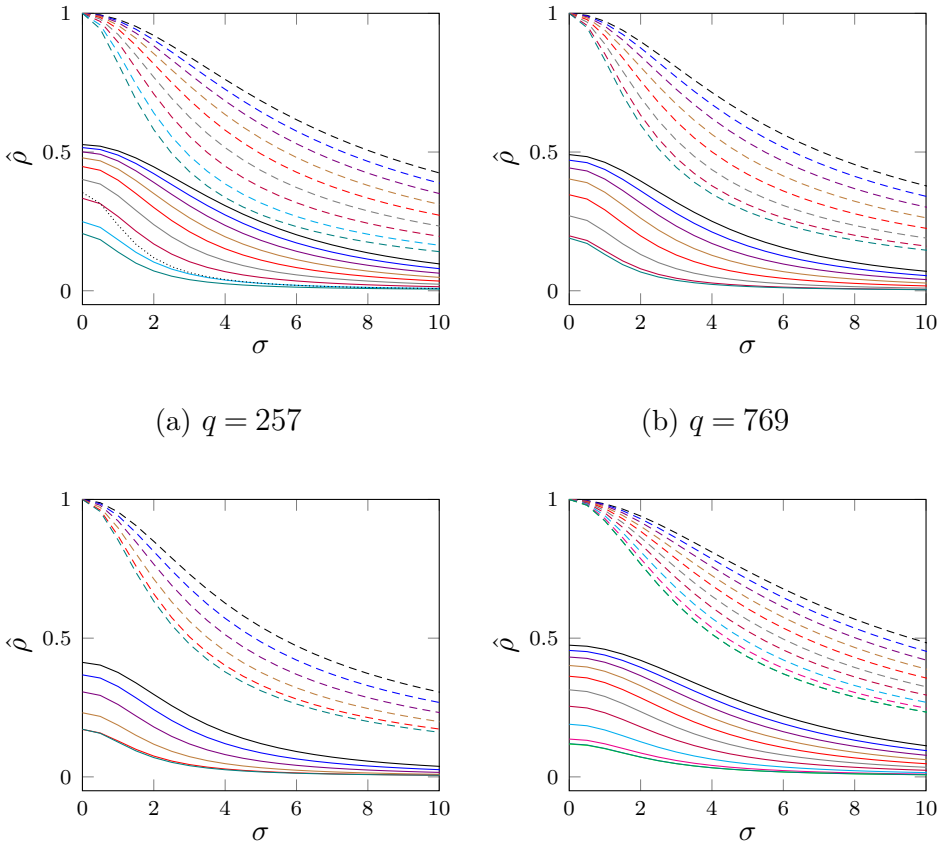
q	257	769	3329	8380417
γ	3.48	4.16	5.19	10.99
γ^-	$\beta - 3.5$	$\beta - 4.17$	$\beta - 5.2$	$\beta - 11$
γ^*	4.5	5.16	6.19	11.99
$\mathsf{E}[\lvert e \rvert]$	1.33	1.4	1.55	2.34

approximately $\gamma + 1$, independent of β and q.

6.1.2. Impact of Signed Arithmetic on Optimal Correlation

To formally assess the impact of signed arithmetic on SCA leakages, we compare the optimal correlation achieved by the state-of-the-art combination function, mean-free product, between the cases when the modular reduction is central and when it is non-central. For consistency and comparison purposes, we also provide correlation results for the unprotected scenario. Figure 6.3 presents the estimated (optimal) correlation for distinct prime q and machine word size β and reduction scenarios. It can be seen that $\hat{\rho}_{opt}$ estimated for central reduction achieves more than twice of the one estimated for the non-central reduction, particularly for $\beta = 16$ and $\beta = 32$. Indeed, $\hat{\rho}_{opt}$ is an increasing function of β for a given q when the reduction is central, complying with our initial observation in Table 6.1. As β decreases, $\hat{\rho}_{opt}$ for the central reduction reaches that of the non-central case as the limit. More importantly, the correlation shows a strong resistance to noise for the signed case. For instance, when $\sigma = 5$ and $q = 257$ and $\beta = 16$ (such as a masked software implementation of Dilithium), $\hat{\rho}_{opt}$ reaches $\hat{\rho} = 0.24$ while we observe $\hat{\rho} = 0.017$ for the unsigned case. We should refer to Table 3.4 showing that β increases the input range of the reduction algorithms. However, our analysis shows that it further increases the associated SCA leakages. It also makes sense to compare the optimal correlation achieved in case of Boolean masking with the other results. Similar to what presented by Prouff et al. (2009), the correlation for Boolean masking reaches $\hat{\rho} = 0.35$ for an 8-bit implementation in a noiseless scenario. Similar to the non-central reduction case, this drops rapidly with the noise, *e.g.* to $\hat{\rho} = 0.02$ for $\sigma = 5$.

The same pattern is being observed in the unprotected scenario. Notably, the correlation is significantly greater when the reduction is central beyond certain values of σ. In particular for $\sigma = 4$ and $q = 3329$, the correlation reaches $\hat{\rho} = 0.62$ with the signed representation whereas it is $\hat{\rho} = 0.37$ with the unsigned representation. However, it is important to highlight that, for the same level of σ and q, the central reduction increases the optimal correlation by $\approx 64\times$. More generally, as illustrated in Figure 6.3, the impact of signed arithmetic on SCA leakage is more prominent in the protected case. An interesting result is that the optimal correlation with the central reduction is relatively close to the correlation observed in unprotected case with non-central reduction, depending on the margin $\mathsf{M}(q, \beta)$. Specifically, for



(a) $q = 257$ (b) $q = 769$

(c) $q = 3329$ (d) $q = 8380417$

Figure 6.3 Estimated correlation with respect to the noise standard deviation σ for different q, β, reduction ranges and protection scenarios. For the first-order masked case, estimations are performed with 1 million samples uniformly taken for $X^{\{0\}}$ and $X^{\{1\}}$ and optimal correlation $\hat{\rho}_{opt}$ is reported in y-axis. Otherwise, estimations are performed with the same number of samples uniformly taken for X and $\hat{\rho}(\mathsf{HW}_\beta(X), \mathsf{L}(X))$ is reported.

- ★ Masked (solid), unprotected (dashed).
- ★ Reduction to $[-q/2, q/2]$ for $q = 257$, $q = 769$ and $q = 3329$: $\beta = 16$ (black), $\beta = 15$ (blue), $\beta = 14$ (violet), $\beta = 13$ (brown), $\beta = 12$ (red), $\beta = 11$ (gray), $\beta = 10$ (purple), $\beta = 9$ (cyan).
- ★ Reduction to $[-q/2, q/2]$ for $q = 8380417$: $\beta = 32$ (black), $\beta = 31$ (blue), $\beta = 30$ (violet), $\beta = 29$ (brown), $\beta = 28$ (red), $\beta = 27$ (gray), $\beta = 26$ (purple), $\beta = 25$ (cyan), $\beta = 24$ (magenta), $\beta = 23$ (green).
- ★ Reduction to $[0, q)$ (teal).
- ★ Boolean masking only in (a) for $\beta = 8$ and $q = 256$ (dotted).

$q = 257$ and $\sigma \geq 3$, the difference in correlation between the masked unsigned and unprotected signed cases is less than 0.05. This leads to the conclusion that the effect of masking can diminish when central reduction is applied.

We replicated the same analysis for the absolute difference combination function C_{abs} (Joye et al., 2005), as detailed in Figure 6.14 (in Section 6.4). Although the

estimated optimal correlations are slightly lower across all values of q, β, and σ, the impact of β and σ follows the same pattern observed for the mean-free product.

For this evaluation, we computed $\mathsf{F}^2_{\text{opt}}$ as a look-up table for each case. The procedure for generating this table is detailed in Algorithm 12. This has a time complexity of $O(q^2)$ to create $\mathsf{F}^2_{\text{opt}}$, which can be costly for large q, such as $q = 8380417$. In Section 6.2, we deal with explicit formulas for $\mathsf{F}^2_{\text{opt}}$ dedicated to central reduction.

Algorithm 12 Computation of $\mathsf{F}^2_{\text{opt}}$ as a look-up table.

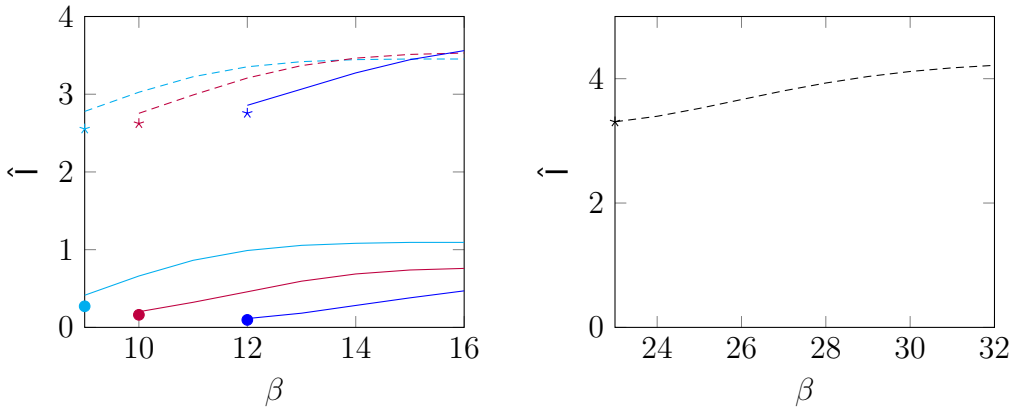
Input: Modulus q
Input: Predicted Leakage Function L'
Output: Look-up table $\mathtt{F}$ representing the function $\mathsf{F}^2_{\text{opt}}$

1: **for** $x = 0$ to $q-1$ **do**
2: $\quad \mathtt{F}_{[x]} = 0$
3: $\quad$ **for** $x_0 = 0$ to $q-1$ **do** ▷ or $x_0 = -q/2$ to $q/2$
4: $\quad\quad x_1 = x - x_0 \pmod q$ ▷ $\text{mod}^+ q$ or $\text{mod}^\pm q$
5: $\quad\quad \mathtt{F}_{[x]} = \mathtt{F}_{[x]} + \mathsf{L}'(x_0) \cdot \mathsf{L}'(x_1)$
6: $\quad$ **end for**
7: $\quad \mathtt{F}_{[x]} = (\mathtt{F}_{[x]}/q) - \left(\frac{\sum_{i=0}^{q-1}(\mathsf{L}'(i))}{q}\right)^2$ ▷ or $\left(\frac{\sum_{i=-q/2}^{q/2}(\mathsf{L}'(i))}{q}\right)^2$, this step can be omitted
8: **end for**
9: **return** $\mathtt{F}$

6.1.3. Information Theoretic Analysis

Additionally, we investigate the MI between X and the HW leakage. In particular, we compute $\hat{\mathsf{I}}(X, \mathsf{HW}_\beta(X))$, as introduced in Section 3.5.4.2, for the unprotected case. For the first-order masked case, we compute the multivariate MI presented in Section 3.5.6.2, $\hat{\mathsf{I}}\big(X, \mathsf{HW}_\beta(X^{\{0\}}), \mathsf{HW}_\beta(X^{\{1\}})\big)$. Figure 6.4 presents numerical results for the MI, across different values of q, β and reduction scenarios[1]. Aligning with our observations from the previous section, MI between X and the HW is higher when the signed representation is employed, which depends on the gap $\mathsf{M}(q, \beta)$. This result holds true for both masked and unprotected cases. In the unprotected scenario, the increase in MI is nearly 1, reflecting the leakage introduced by the sign. On the other hand, with masking, the increase in MI is up to $\approx 4\times$ with β. Recall that the correlation-based analysis also indicates that negative impact of central reduction is more pronounced in the masked case. Recall that, MIA (Gierlichs et al., 2008)

[1] MI results for $q = 8380417$ with masking enabled are not provided due to the computational complexity of the experiments. Nevertheless, they can be inferred from the results of the other primes studied.



(a) $q = 257$, $q = 769$, $q = 3329$ (b) $q = 8380417$

Figure 6.4 Mutual information for different q, β, reduction ranges, and protection scenarios. For the unprotected case, $\hat{\mathsf{I}}(X, \mathsf{HW}_\beta(X))$ is reported while for the first-order masked case $\hat{\mathsf{I}}\big(X, \mathsf{HW}_\beta(X^{\{0\}}), \mathsf{HW}_\beta(X^{\{1\}})\big)$ is reported in y-axis.

- ★ Reduction to $[-q/2, q/2]$: masked (solid), unprotected (dashed). Reduction to $[0, q)$: masked (circle), unprotected (star).
- ★ Moduli: $q = 257$ (cyan), $q = 769$ (purple), $q = 3329$ (blue). $q = 8380417$ (black).

is a generic distinguisher where the attacker is not required to predict the device's leakage model. Given that the leakage model in our case is linear with HW as defined in Equation (6.1), and results with MIA are consistent with our previous findings, we stick with CPA for the remainder of the chapter.

6.2. Absolute Value Prediction Function

When the target operation is protected using Boolean masking and the underlying circuit is a noisy Hamming weight of intermediates (as in Equation (6.1)), it is shown by Prouff et al. (2009) that HW_β can be effectively used as the optimal prediction function for HOCPA attacks. However, this is not necessarily the case when masking is arithmetic. To provide an intuition, we estimated $\hat{\rho}(\mathsf{HW}_\beta(X), \mathsf{F}^2_{\text{opt}}(X))$, as proposed by Prouff et al. (2009), to measure the accuracy of prediction functions for the studied adversary model, presented in Table 6.2. The results indicate a significant correlation loss in all scenarios. Hence, in this section, we search for an explicit formula for the optimal prediction function in case of arithmetic masking. Precisely, we show that the absolute value function can be used as the optimal prediction function when targeting arithmetic masking where central reduction is employed.

Table 6.2 Estimated correlation between the Hamming Weight and optimal prediction functions, $\hat{\rho}(\mathsf{HW}_\beta(X), \mathsf{F}^2_{\text{opt}}(X))$, for different moduli q and β. The estimations are performed with 1 million uniformly random samples for X while the reduction is central.

β \ q	257	769	3329
16	**−0.741**	**−0.732**	**−0.661**
15	−0.723	−0.706	−0.607
14	−0.698	−0.671	−0.527
13	−0.665	−0.621	−0.405
12	−0.618	−0.545	−0.223
11	−0.548	−0.427	
10	−0.442	−0.241	
9	−0.273		

β \ q	8380417
32	**−0.706**
31	−0.684
30	−0.655
29	−0.618
28	−0.570
27	−0.507
26	−0.424
25	−0.314
24	−0.173
23	−0.003

6.2.1. Distribution of the Secret Knowing the Sign of Shares

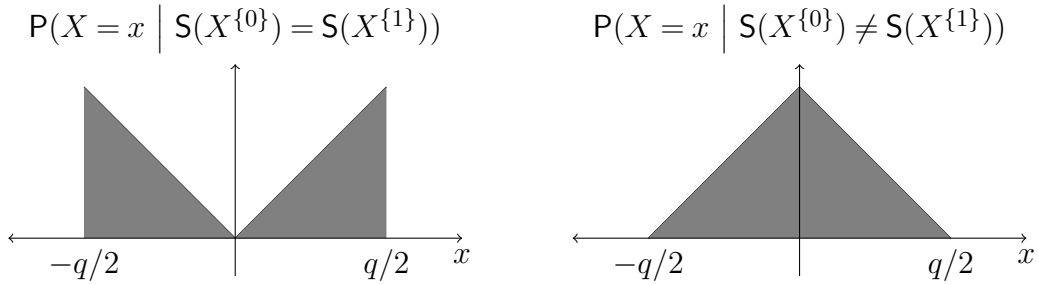
Consider two uniformly random variables $X^{\{0\}}, X^{\{1\}} \in^{\pm} \mathbb{Z}_q$ and their modular addition $X^{\{0\}} + X^{\{1\}} \bmod^{\pm} q$. Given the signs of both variables, Figure 6.5 demonstrates the probability distribution of $X^{\{0\}} + X^{\{1\}} \bmod^{\pm} q$. As depicted in the figure, there are two cases for the distribution given the sign of both random variables. If the sign of $X^{\{0\}}$ and $X^{\{1\}}$ are the same, namely $\mathsf{S}(X^{\{0\}}) = \mathsf{S}(X^{\{1\}})$, then the probability is distributed around $\pm q/2$. Otherwise, it is centered around 0. Indeed, the distributions correspond to the convolution of probability distribution functions. More precisely, one of

$$\mathsf{P}(X^{\{0\}} = x^{\{0\}} \mid X^{\{0\}} < 0), \quad \mathsf{P}(X^{\{0\}} = x^{\{0\}} \mid X^{\{0\}} \geq 0)$$

is convoluted to one of

$$\mathsf{P}(X^{\{1\}} = x^{\{1\}} \mid X^{\{1\}} < 0), \quad \mathsf{P}(X^{\{1\}} = x^{\{1\}} \mid X^{\{1\}} \geq 0).$$

Now suppose that $X^{\{0\}}$ and $X^{\{1\}}$ are arithmetic shares representing a secret intermediate variable $X = X^{\{0\}} + X^{\{1\}}$. Then, the above discussion shows that information about the sign of the individual shares leads to a strong effect on the distribution of the secret X.



(a) Case 1: $(X^{\{0\}} < 0$ and $X^{\{1\}} < 0)$ or $(X^{\{0\}} \geq 0$ and $X^{\{1\}} \geq 0)$ *i.e.* $\mathsf{S}(X^{\{0\}}) = \mathsf{S}(X^{\{1\}})$

(b) Case 2: $(X^{\{0\}} \geq 0$ and $X^{\{1\}} < 0)$ or $(X^{\{0\}} \geq 0$ and $X^{\{1\}} < 0)$ *i.e.* $\mathsf{S}(X^{\{0\}}) \neq \mathsf{S}(X^{\{1\}})$

Figure 6.5 Probability distributions of $X = X^{\{0\}} + X^{\{1\}} \bmod^{\pm} q$ for $X^{\{0\}}, X^{\{1\}} \in^{\pm} \mathbb{Z}_q$

6.2.2. A Model for Mean-Free Product

In the previous section, we showed that the HW of 2's complement representation of an integer in $[-q/2, q/2]$ is a noisy indicator of its sign. Also, recall the leakage in CMOS circuits which is highly relevant to the HW of processed data (see Equation (6.1)). Now, let $L_0 = \mathsf{L}(X^{\{1\}})$ and $L_1 = \mathsf{L}(X^{\{1\}})$ denote the leakage associated to the random shares $X^{\{0\}}$ and $X^{\{1\}}$. The mean-free product can be written as follows.

$$\mathsf{C}(L_0, L_1) = \Big(\alpha_0 \cdot \mathsf{HW}_\beta(X^0) + \mathsf{N}(\mu_0, \sigma_0) - \mathsf{E}\Big[\alpha_0 \cdot \mathsf{HW}_\beta(X^0) + \mathsf{N}(\mu_0, \sigma_0)\Big]\Big) \cdot \tag{6.4}$$

$$\Big(\alpha_1 \cdot \mathsf{HW}_\beta(X^1) + \mathsf{N}(\mu_1, \sigma_1) - \mathsf{E}\Big[\alpha_1 \cdot \mathsf{HW}_\beta(X^1) + \mathsf{N}(\mu_1, \sigma_1)\Big]\Big) \tag{6.5}$$

For the sake of simplicity, we assume $\alpha = \alpha_0 = \alpha_1$, $\mu = \mu_0 = \mu_1$, and $\sigma = \sigma_0 = \sigma_1$. As $X^{\{0\}}$ and $X^{\{1\}}$ are uniformly random signed integers in $^{\pm}\mathbb{Z}_q$ and represented by β bits in the computer memory, $\mathsf{E}\Big[X^{\{0\}}\Big] = \mathsf{E}\Big[X^{\{1\}}\Big] = \beta/2$. Then, we can write

$$\mathsf{C}(L_0, L_1) = \Big(\alpha \cdot \mathsf{HW}_\beta(X^0) + \mathsf{N}(\mu, \sigma) - (\alpha \cdot \beta/2 + \mu)\Big) \cdot \tag{6.6}$$

$$\Big(\alpha \cdot \mathsf{HW}_\beta(X^1) + \mathsf{N}(\mu, \sigma) - (\alpha \cdot \beta/2 + \mu)\Big) \tag{6.7}$$

and by distributing the terms and as $\mathsf{N}(0, \sigma) = \alpha \cdot \mathsf{N}(0, \sigma/\alpha)$,

(6.8)

$$\mathsf{C}(L_0, L_1) = \alpha^2 \Big(\mathsf{HW}_\beta(X^0) - \beta/2 + \mathsf{N}(0, \sigma/\alpha)\Big) \cdot \Big(\mathsf{HW}_\beta(X^1) - \beta/2 + \mathsf{N}(0, \sigma/\alpha)\Big)$$

By plugging Equation (6.2) into Equation (6.8) we have

(6.9)
$$\mathsf{C}(L_0, L_1) = \alpha^2 \Big(\mathsf{S}(X^{\{0\}}) \cdot (\gamma - \beta/2) + \Big(1 - \mathsf{S}(X^{\{0\}})\Big) \cdot (\beta/2 - \gamma) + e_0 + \mathsf{N}(0, \sigma/\alpha) \Big) \cdot$$

(6.10)
$$\Big(\mathsf{S}(X^{\{1\}}) \cdot (\gamma - \beta/2) + \Big(1 - \mathsf{S}(X^{\{1\}})\Big) \cdot (\beta/2 - \gamma) + e_1 + \mathsf{N}(0, \sigma/\alpha) \Big)$$

6.2.3. Conditional Probability of Sign Equality

As explained above and shown by Figure 6.5, we conclude that

(6.11)
$$\mathsf{P}\Big(X = x \;\Big|\; \mathsf{S}(X^{\{0\}}) = \mathsf{S}(X^{\{1\}}) \Big) = (2/q) \cdot \Big(|x| / (q/2) \Big)$$

(6.12)
$$= |x| \cdot (4/q^2).$$

Based on the dependency of X on $\mathsf{S}(X^{\{0\}})$ and $\mathsf{S}(X^{\{1\}})$, we show that the conditional probability $\mathsf{P}\Big(\mathsf{S}(X^{\{0\}}) = \mathsf{S}(X^{\{1\}}) \;\Big|\; X = x \Big)$ is a multiple of $|x|$.

(6.13)
$$\mathsf{P}\Big(\mathsf{S}(X^{\{0\}}) = \mathsf{S}(X^{\{1\}}) \;\Big|\; X = x \Big) =$$

(6.14)
$$\frac{\mathsf{P}\Big(X = x \;\Big|\; \mathsf{S}(X^{\{0\}}) = \mathsf{S}(X^{\{1\}}) \Big) \cdot \mathsf{P}\Big(\mathsf{S}(X^{\{0\}}) = \mathsf{S}(X^{\{1\}}) \Big)}{\mathsf{P}\Big(X = x \Big)}$$

(6.15)
$$= \frac{|x| \cdot (4/q^2) \cdot 1/2}{1/q} = (2/q) \cdot |x|$$

6.2.4. Estimating the Optimal Prediction Function

We make use of the conditional probability to formally estimate $\mathsf{E}\Big[\mathsf{C}\,(L_0, L_1) \Big| X = x \Big]$ as

$$\mathsf{E}\left[\mathsf{C}(L_0,L_1)\middle|X=x,\mathsf{S}\left(X^{\{0\}}\right)=\mathsf{S}\left(X^{\{1\}}\right)\right]\cdot \tag{6.16}$$

$$\mathsf{P}\left(\mathsf{S}\left(X^{\{0\}}\right)=\mathsf{S}\left(X^{\{1\}}\right)\middle|X=x\right)+ \tag{6.17}$$

$$\mathsf{E}\left[\mathsf{C}(L_0,L_1)\middle|X=x,\mathsf{S}\left(X^{\{0\}}\right)\neq\mathsf{S}\left(X^{\{1\}}\right)\right] \tag{6.18}$$

$$\mathsf{P}\left(\mathsf{S}\left(X^{\{0\}}\right)\neq\mathsf{S}\left(X^{\{1\}}\right)\middle|X=x\right) \tag{6.19}$$

Considering the terms including e_0 and e_1 as error, we can write $\mathsf{E}\left[\mathsf{C}(L_0,L_1)\middle|X=x\right]$ as

(6.20)

$$\mathsf{E}\left[\mathsf{C}(L_0,L_1)\middle|X=x\right]=(\gamma-\beta/2)^2\cdot(2/q)\cdot|x|\ (\gamma-\beta/2)\cdot(\beta/2-\gamma)\cdot(1-2/q\cdot|x|)+e_{\mathsf{C}}$$

$$=(\gamma-\beta/2)^2\cdot(4/q)\cdot|x|-(\gamma-\beta/2)^2+e_{\mathsf{C}} \tag{6.21}$$

where

$$e_{\mathsf{C}}=\mathsf{E}\left[e_0\cdot e_1\ \middle|\ X=x\right] \tag{6.22}$$

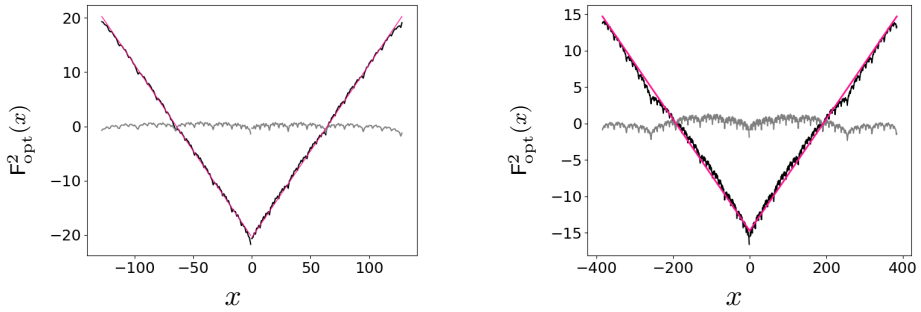
$$+2\mathsf{E}\left[e_1\cdot\mathsf{S}\left(X^{\{0\}}\right)\cdot(\gamma-\beta/2)+\left(1-\mathsf{S}\left(X^{\{0\}}\right)\right)\cdot(\beta/2-\gamma)\mid X=x\right] \tag{6.23}$$

$$=\mathsf{E}\left[e_0\cdot e_1\mid X=x\right]+4(\gamma-\beta/2)\cdot\mathsf{E}\left[e_0\cdot\mathsf{S}\left(X^{\{1\}}\right)\mid X=x\right] \tag{6.24}$$

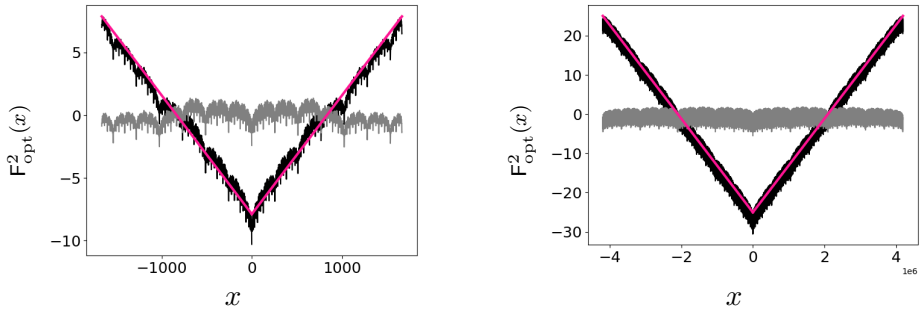
Note that e_{C} can be derived for any valid β when $E\left[e_0\cdot e_1\ \middle|\ X=x\right]$ and $E\left[e_0\cdot\mathsf{S}(X^{\{1\}})\ \middle|\ X=x\right]$ are pre-computed. This approach is beneficial particularly if q is large (*e.g.* 8380417) and the attacker does not know β in advance. Intuitively, e_{C} is relatively small, and its impact decreases as the margin $\mathsf{M}(q,\beta)$ increases. With sufficient $\mathsf{M}(q,\beta)$, Equation (6.21) is accurately approximated by

$$E\left[\mathsf{C}(L_0,L_1)\ \middle|\ X=x\right]\approx c_0\cdot|x|+c_1 \tag{6.25}$$

where $c_0=(\gamma-\beta/2)^2\cdot4/q$ and $c_1=-(\gamma-\beta/2)^2$. Figure 7.1 visualizes these estimations and the corresponding error for two distinct cases of q and β. Since constants c_0 and c_1 do not affect the correlation, $\mathsf{F}_{\text{abs}}(x)=|x|$ can be used as the optimal prediction function.



(a) $q = 257$, $\beta = 16$ (b) $q = 769$, $\beta = 16$



(c) $q = 3329$, $\beta = 16$ (d) $q = 8380417$, $\beta = 32$

Figure 6.6 Visualization of optimal prediction function for the central reduction range $[-q/2, q/2]$ for q and β. $\mathsf{F}^2_{\mathrm{opt}}(x)$ (black), e_{C} (gray), $c_0 \cdot |x| + c_1$ (pink).

6.2.5. On the Accuracy of $\mathsf{F}_{\mathrm{abs}}$

Since Equation (6.24) does not allow to give an exact explicit formula for e_{C}, we proposed approximating $\mathsf{F}^2_{\mathrm{opt}}(X)$ using the absolute value function. In order to evaluate the accuracy loss as a result of using $\mathsf{F}_{\mathrm{abs}}$ instead of Equation (6.21), we estimated $\hat{\rho}\left(\mathsf{F}_{\mathrm{abs}}(X), \mathsf{F}^2_{\mathrm{opt}}(X)\right)$. Table 6.3 presents the estimations, indicating $\hat{\rho} > 0.99$ for all studied q and β couples mentioned in Section 6.1. Therefore, we conclude that the absolute value prediction function is highly accurate for the attacks we consider in this work. However, for any other settings (q and β couples), where the absolute value prediction function leads to an undesired accuracy, the $\mathsf{F}^2_{\mathrm{opt}}(X)$ function can be computed for all possible values of X as $E\left[\mathsf{C}(L_0, L_1) \middle| X\right]$ (*e.g.* Equation (6.21)).

The $\mathsf{F}_{\mathrm{abs}}$ can also be used with the absolute difference combination function $\mathsf{C}_{\mathrm{abs}}$. The corresponding results for this configuration are presented in Section 6.4.

Table 6.3 Estimated correlation between the absolute value and optimal prediction functions, $\hat{\rho}(\mathsf{F}_{\text{abs}}(X), \mathsf{F}^2_{\text{opt}}(X))$, for different moduli q and β. The estimations are performed with 1 million uniformly random samples for X while the reduction is central.

β \ q	257	769	3329
16	**0.999**	**0.997**	**0.992**
15	0.999	0.996	0.984
14	0.998	0.993	0.961
13	0.997	0.987	0.877
12	0.994	0.970	0.546
11	0.985	0.908	
10	0.956	0.635	
9	0.825		

β \ q	8380417
32	**0.998**
31	0.998
30	0.997
29	0.995
28	0.992
27	0.983
26	0.961
25	0.892
24	0.659
23	0.214

6.2.6. Alternative Prediction Function

As the chosen target function is a multiplication for this study, one can fine-tune the optimal prediction function by considering zero-value public data, *i.e.* $\hat{c}_i = 0$. Notice that for the complete NTT, $X = 0$ if and only if $\hat{c}_i = 0$ (assuming $\hat{s}_i \neq 0$), because of the specialty of 0 in multiplication. Then, $X^{\{0\}}$ and $X^{\{1\}}$ become 0 as $X^{\{j\}} = \hat{s}_i^j \cdot 0$ and $\mathsf{F}^2_{\text{opt}}(0) = \mathsf{E}[\mathsf{C}(Y_0, Y_1) | X^{\{0\}} = 0, X^{\{1\}} = 0] = \alpha^2 \cdot (\mathsf{E}[\mathsf{W}_\beta(X) + \mathsf{N}(\mu, \sigma)])^2$, see Equation (6.1). When the NTT is incomplete, X can be 0 even though $\hat{c}_i$ is non-zero, because the target X is the addition of two multiplications which can sum up to 0. Recall that X is a function of $\hat{r}_{[i][j]}$ in this case, which is formulated in Equation (3.35) and Equation (3.36). However, since having $\hat{c}_i = 0$ is less likely for the incomplete NTT ($1/q^2$ for $\hat{c}_{i,0} = \hat{c}_{i,1} = 0$, assuming uniformly random $\hat{c}_i$) we do not concentrate on this case.

To take the advantage of zero-value public data, we define the alternative absolute value prediction function as follows.

$$\mathsf{F}^*_{\text{abs}}(X) = \begin{cases} \big((\beta/2)^2 - c_1\big)/c_0, & \text{if } X = 0 \\ |X| & \text{otherwise} \end{cases} \tag{6.26}$$

Recall that $\mathsf{E}\Big[\mathsf{C}(L_0, L_1) \Big| X^{\{0\}} = 0, X^{\{1\}} = 0\Big] = (\beta/2)^2$.

6.3. Results

We provide experimental results through simulations and real-device traces.

6.3.1. Simulation Results

In this section, we present the result of HOCPA attacks explained in Chapter 4 making use of simulated traces. In our simulations, we consider different noise levels and various reduction scenarios for each of the studied q. We particularly compare the required number of traces with and without central reduction. Needless to say that F_{abs} and $\mathsf{F}^{*}_{\text{abs}}$ are used as the prediction function when the reduction is central while $\mathsf{F}^{2}_{\text{opt}}$ is computed as a look-up table otherwise (as done in Section 6.1.2).

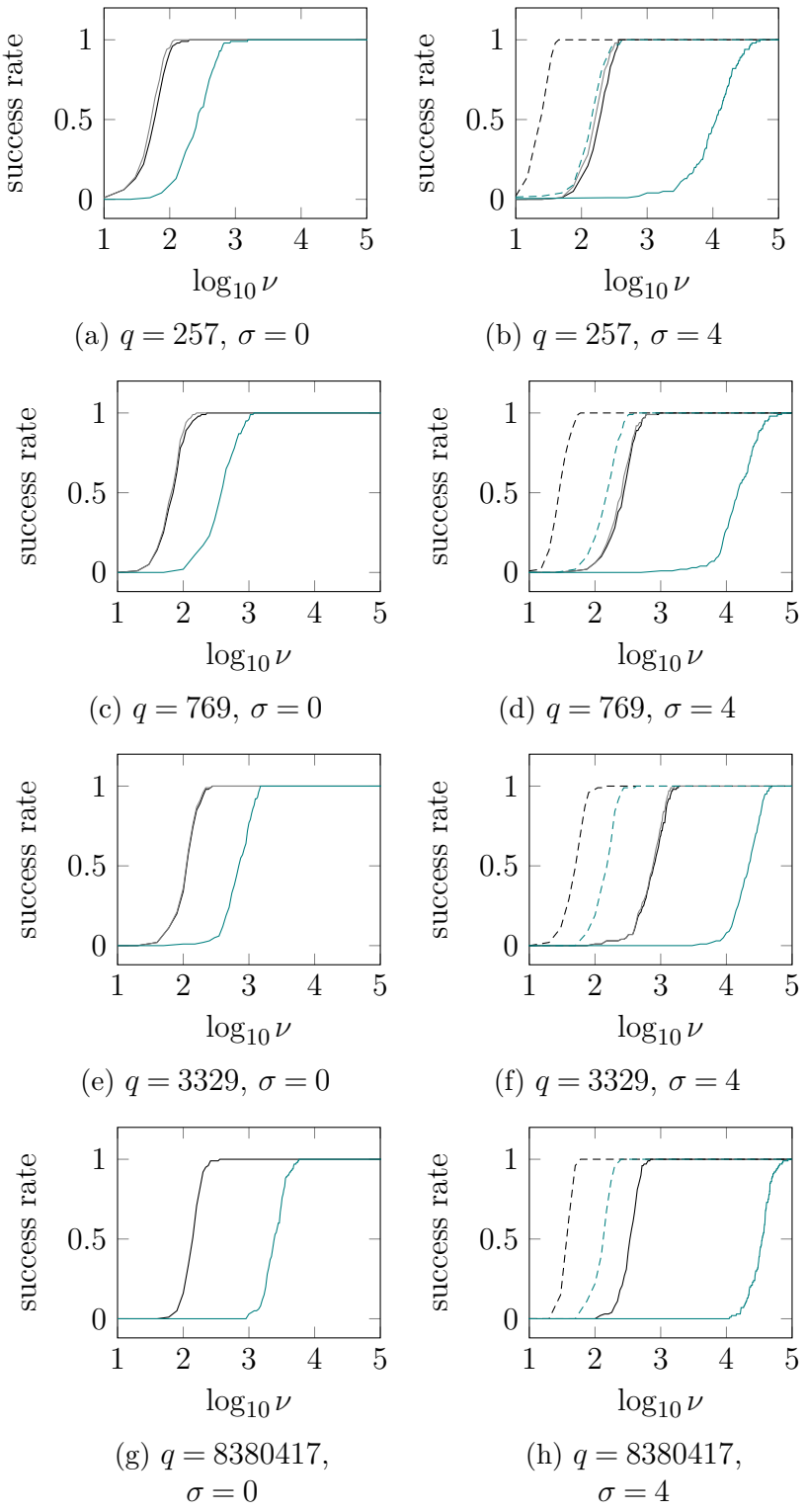
We generated simulated traces for the base multiplication using Algorithm 11 explained in Section 4.7. We set $\mathsf{L} = \mathsf{HW}_{\beta}$ since we consider Hamming Weight leakage. The masking order was set to $d = 2$ since we evaluate first-order masking, while $d = 1$ was also considered as a reference to assess the impact of masking. For all simulations, we used $m = 100$, which defines the number of sub-keys, and $\mu = 0$. Recall from Section 3.3 that while $q = 8380417$ allows a complete NTT with the ring dimension $n = 256$, other moduli $q = 257$, $q = 769$, and $q = 3329$ do not allow complete NTT but allow an incomplete NTT. However, the simulations aim to benchmark our introduced absolute value prediction function and compare leakage of different reduction schemes. Therefore, there is no harm in doing the simulations as if the NTT is complete ($f_1 = 0$ in Algorithm 11), which only affects the number of hypotheses. The comparison between the hypothetical and observed leakages is not affected by this behavior.

Figure 7.2 presents the corresponding results, with respect to q, ν, and σ, while the success rate refers to $(\#\ \text{correctly predicted}\ \mathbf{s}_{[i]}/m)$. It should be noted that the number of traces is displayed on a logarithmic scale. As evident by the results, the implementations with central reduction are significantly more vulnerable to these non-profiled HOCPA attacks in terms of the number of traces. For instance, when $q = 3329$ and $\sigma = 0$, the attack against non-central reduction needs 1500 traces to succeed, which is $6\times$ more than the number of traces needed when the reduction is central. Moreover, the noise σ has a greater impact on the attacks on implementations with non-central reduction compared to the central case. When $q = 3329$ and $\sigma = 4$, the

attack on non-central reduction needs 45 k traces to succeed, which is 31× more than what is required in case of central reduction. The difference with respect to the number of traces reaches 123× for $q = 257$ and $\sigma = 4$. We conclude that HOCPA with F_{abs} targeting central reduction remains a major threat in different noise levels conforming with the observation shown in Figure 6.3. Based on the aforementioned decrease in the number of traces required to attack, employing central reduction in masked LBC might be not the best choice from the SCA perspective. Although central reduction is sometimes preferred for efficiency purposes, non-central reduction can harden higher-order SCA attacks in security-demanding applications.

In general, the attack on central reduction needs relatively small number of traces to succeed. However, the number of traces for a successful attack depends on the margin $\mathsf{M}(q, \beta)$, which is slightly worse for $q = 3329$ compared to the other primes considered here. For $q = 3329$ and $\sigma = 4$, HOCPA with $\mathsf{F}^*_{\text{abs}}$ requires only 1400 traces to succeed. In a noiseless scenario, where $q = 257$ and $\sigma = 0$, the attack only needs around 110 traces. As previously stated, we consider only two cases $\beta = 16$ and $\beta = 32$. However, we anticipate from Figure 6.3 that, as β decreases, the number of required traces to attack the central reduction schemes gets closer to that when a non-central reduction scheme is employed. As anticipated, the advantage of using $\mathsf{F}^*_{\text{abs}}$ over F_{abs} highly depends on the number of times when $\mathbf{c}_i = 0$. For instance when $q = 257$ and $\sigma = 4$, $\mathsf{F}^*_{\text{abs}}$ leads to %13 reduction in the number of required traces in our experiments. As the chance of observing zero-values decreases when q increases, the advantage of $\mathsf{F}^*_{\text{abs}}$ decreases as well. For example, when $q = 3329$ and $\sigma = 4$, $\mathsf{F}^*_{\text{abs}}$ reduces ν by %9 compared to F_{abs}.

The disadvantage of signed arithmetic is also evident in the unprotected but noisy scenario, where the number of required traces increases by up to two orders of magnitude. For example, when $q = 257$ and $\sigma = 4$, the number of traces required with the central reduction is 9.5× higher than the non-central reduction case. Notice that the increase in ν when using signed arithmetic is smaller compared to the protected case, by a factor of 13× for $q = 257$. Consequently, the simulation results align with our observation in Section 6.1, indicating that the negative impact of signed arithmetic is more significant in masked schemes. Also observe the statistical results for first-order attacks on unsigned arithmetic and second-order attacks on signed arithmetic are closely aligned. In the extreme case of $q = 257$, the number of required traces for these two attacks, despite their different orders, is nearly identical. For $q = 8380417$, the difference in ν for a successful attack between these two scenarios is only 2.8×.



(a) $q = 257$, $\sigma = 0$

(b) $q = 257$, $\sigma = 4$

(c) $q = 769$, $\sigma = 0$

(d) $q = 769$, $\sigma = 4$

(e) $q = 3329$, $\sigma = 0$

(f) $q = 3329$, $\sigma = 4$

(g) $q = 8380417$, $\sigma = 0$

(h) $q = 8380417$, $\sigma = 4$

Figure 6.7 Success rates of first-order and second-order CPA attacks on simulated traces generated with different q, σ and reduction ranges with respect to ν. For each point in the curves, 100 experiments have been performed with random data.

- ★ SCA Protection: first-order masked (solid), unprotected (dashed).
- ★ Reduction ranges: $[-q/2, q/2]$ (black, gray), $[0, q)$ (teal).
- ★ Prediction functions for HOCPA: F_{abs} (black), $\mathsf{F}^*_{\text{abs}}$ (gray), $\mathsf{F}^2_{\text{opt}}$ (teal).

6.3.2. Practical Results: Application to Kyber

In this section, we present the result of applying our proposed approach to perform successful HOCPA attacks on a protected implementation of Kyber. It is important to note that we have previously compared the SCA leakage between central and non-central reduction. Therefore, the motivation of this section is to evaluate the difficulty of conducting successful HOCPA attacks targeting central reduction using real data. Source code of the implementations and the attack scripts as well as the simulations presented in the previous section are publicly available.[2]

6.3.2.1. Target implementation

We focus on the ARM Cortex-M4 specific open-source and first-order masked implementation of Kyber from Bronchain & Cassiers (2022)[3]. The polynomial arithmetic of the implementation is mostly in assembly, ported from the pqm4 project Kannwischer et al. (2019) and employs the Montgomery reduction that we illustrated in Section 3.4. We also created a second version of the victim implementation by integrating the latest iteration of polynomial arithmetic from pqm4 which employs the Plantard reduction based on Huang et al. (2022)[4]. Hereafter, we denote the untouched target implementation with Montgomery reduction by Ψ_M and the in-house version with Plantard reduction by Ψ_P. In particular, we focus on the function `basemul_asm` which implements the base multiplication $\hat{s} \star \hat{c}$ in the incomplete NTT domain for both Montgomery and Plantard versions. We should note that – to the best of our knowledge – all masked implementations of post-quantum algorithms on the ARM Cortex-M4 that have been reported in the literature are built on top of pqm4 by directly porting the linear operations including polynomial arithmetic (Azouaoui et al., 2023; Beirendonck, D'anvers, Karmakar, Balasch & Verbauwhede, 2021; Bos, Gourjon, Renes, Schneider & Van Vredendaal, 2021; Heinz & Dreo Rodosek, 2023; Heinz et al., 2022). Therefore, we believe that assessing the most recent iteration of pqm4, featuring state-of-the-art polynomial arithmetic, would be beneficial. The open-source Kyber implementation (Bronchain & Cassiers, 2022) employs the Montgomery reduction since the more efficient Plantard reduction

[2] `https://github.com/toluntosun21/ExploitingCentralReduction`

[3] `https://github.com/uclcrypto/pqm4_masked/` commit hash: **5fe90ba**

[4] `https://github.com/mupq/pqm4` commit hash: **3743a66**

did not exist when the polynomial implementation was imported from pqm4. Our experiments are centered around the medium security level, *i.e.* Kyber768, though it does not affect our approach and results.

6.3.2.2. Setup

Our measurement setup is presented in item 6.8. We used NewAE ChipWhisperer CW1200 together with the CW308 UFO board to collect power traces. The victim program was running on a STM32F303, which is equipped with an ARM Cortex-M4, as shown in Figure 6.9. The frequency of the core is set to 7.3 MHz by an external reference clock which is also given to the power-collecting facility (analog-to-digital converter) while 4 power samples are recorded at each clock cycle. We provided a trigger signal for the power-collecting module to indicate the beginning of the function `basemul_asm` for the first share. Hence, only the samples related to the base multiplication were recorded. The attacks have been performed using the scared library, with an in-house developed Python model that mimics the intended Kyber implementation, as in Chapter 5. A laptop equipped with an AMD Ryzen™ 7 7840HS[5] 8-core processor and 64 GB RAM was used for running the attack.

6.3.2.3. Attack details

A mean trace over 1000 traces is presented in Figure 7.3. It should be noted that the iterations of the function `basemul_asm` are visible through the mean trace for both shares. In order to reveal each $\hat{s}_{[i]}$, in the corresponding attacks we have only taken into account the relevant part of the power traces based on the iterations. The point-of-interest (PoI) selection follows the method explained in Section 7.2.5.3. We used a constant offset to combine the leakages associated to two shares (by mean-free product as explained in Section 3.5.6.1) based on the pattern observed in Figure 7.3. It is noteworthy to mention that the same strategy can be easily adapted via educated guesses without prior knowledge of the specific implementation. Recall that $\hat{s}_{[i]}$ is a degree-1 polynomial in Kyber, and two coefficients must be predicted together based on the outline presented in Chapter 4. We tested $q \cdot q/2$ hypotheses

[5]https://www.amd.com/en/products/processors/laptop/ryzen/7000-series/amd-ryzen-7-7840hs.html

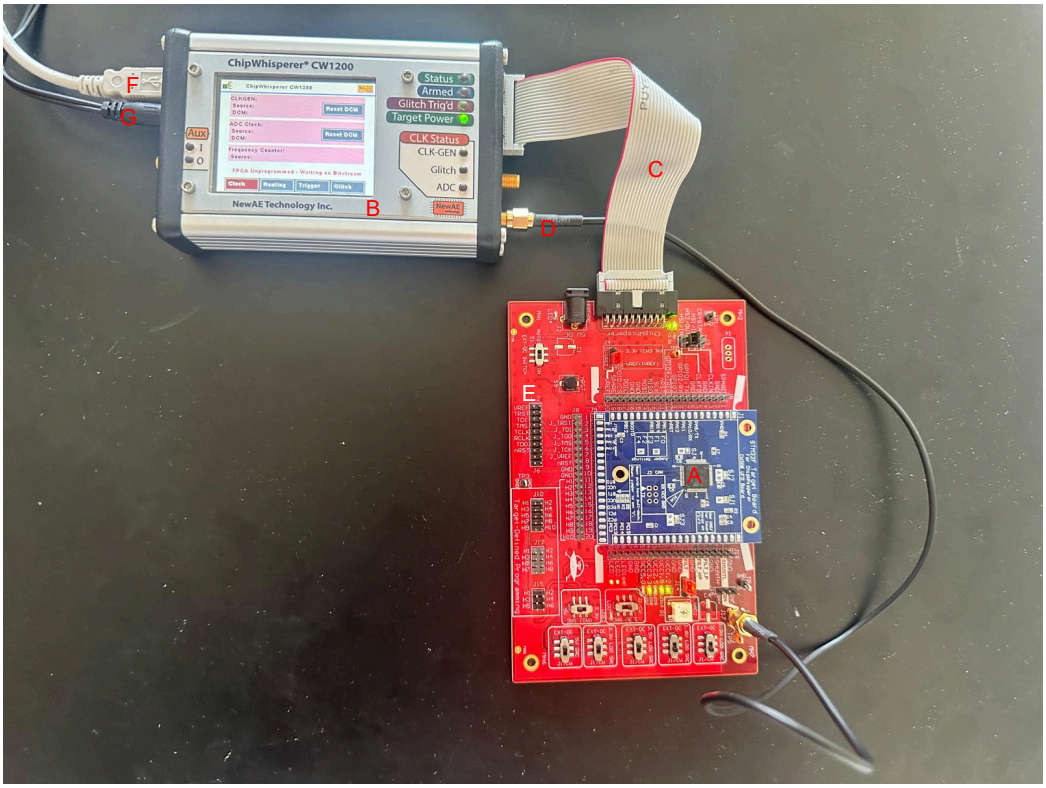


Figure 6.8 ChipWhisperer CW1200 Trace Collection Setup.

(A) Victim microcontroller device.
(B) CW1200 capture hardware, acting as the oscilloscope and handling data transfer.
(C) Programming and I/O data transfer cable.
(D) Power leakage measurement cable.
(E) CW308-UFO motherboard hosting the victim device.
(F) USB cable connected to the host PC.
(G) Power cable for the CW1200.

($\approx 2^{22.4}$ as $q = 3329$) with F_{abs} as the prediction function, so that either the actual secret or its additive inverse is found (for both Ψ_M and Ψ_P). The target of the attack is the higher-degree coefficient of each $\hat{s}_{[i]} \cdot \hat{c}_{[i]}$, precisely $\mathsf{G}(\hat{s}_{[i]}, \hat{c}_{[i]}) = \hat{r}_{[i][1]}$ computed in Equation (3.36) (also see Section 4.3). When F_{abs} is used as the prediction function, a hypothesis and its additive inverse, $\pm\hat{s}_{[i]}$, gets the same correlation score due to the nature of absolute value function.[6]

[6] One option to distinguish the correct hypothesis from its additive inverse is to re-run the same HOCPA attack on two hypotheses $\pm\hat{s}_{[i]}$ using the sign function S, see Equation (6.1.1). This intuition is based on the fact that not every bit of the intermediate values equally contributes in the amount of power consumption, *i.e.* not an ideal HW model. This leads F_{opt} to be not fully symmetric with respect to the y axis, see Figure 7.1.

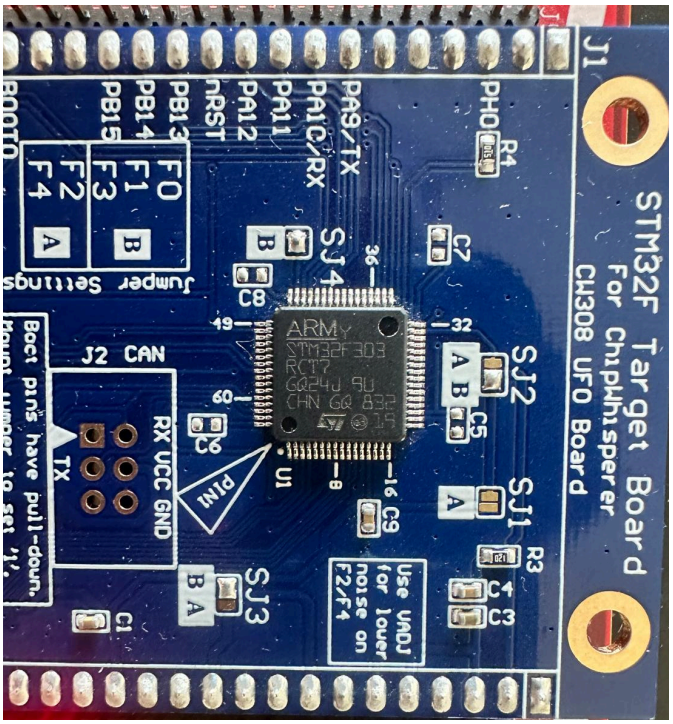
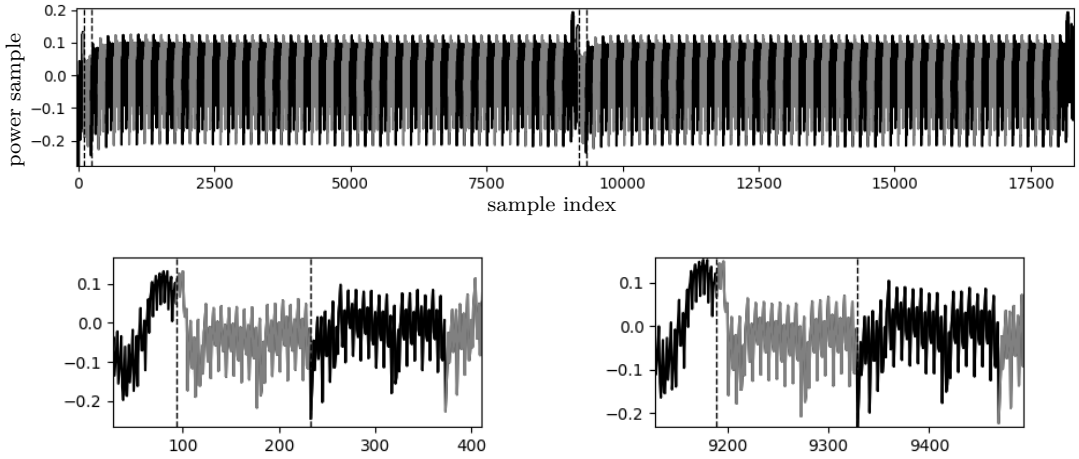
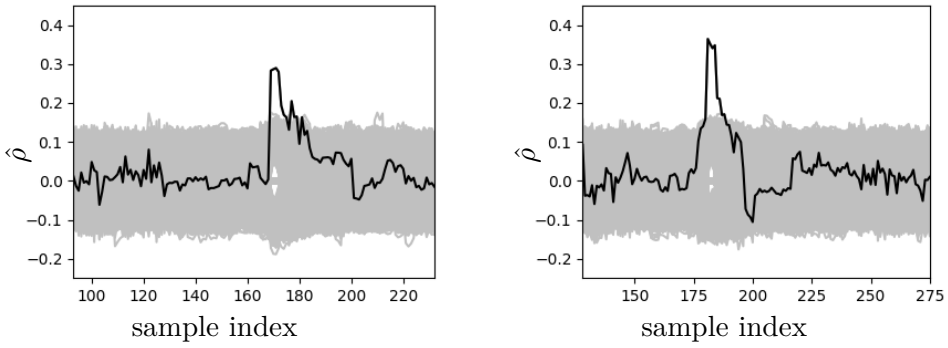


Figure 6.9 Microcontroller STM32F303 running the victim program.



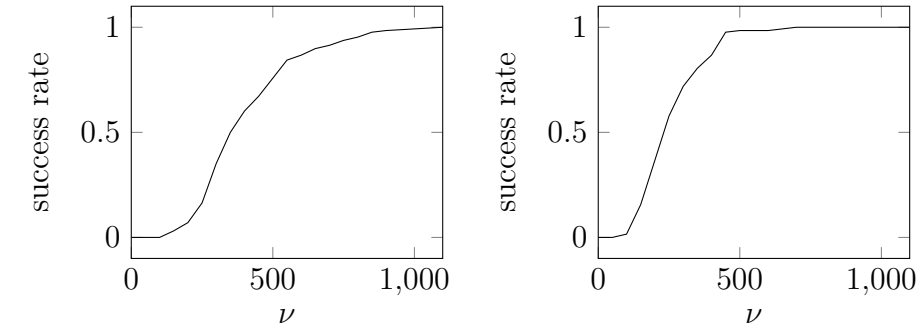
(a) $\hat{s}^{\{0\}}_{[0]} \cdot \hat{c}_{[0]}$ and $\hat{s}^{\{0\}}_{[1]} \cdot \hat{c}_{[1]}$ (b) $\hat{s}^{\{1\}}_{[0]} \cdot \hat{c}_{[0]}$ and $\hat{s}^{\{1\}}_{[1]} \cdot \hat{c}_{[1]}$

Figure 6.10 The mean power trace associated with the execution of the base multiplication function `basemul_asm` in Ψ_M for both shares, $\hat{s}^{\{0\}} \star \hat{c}$ and $\hat{s}^{\{1\}} \star \hat{c}$. The iterations of the function are marked by interleaving black and gray colors. Due to loop unrolling, 64 iterations are observed for each share instead of $n/2 = 128$. Recall that the ring dimension $n = 256$ for Kyber and 7-layer NTT is performed. The first iterations of `basemul_asm` for both shares are marked and zoomed in (a) and (b). The same observation applies to Ψ_P.



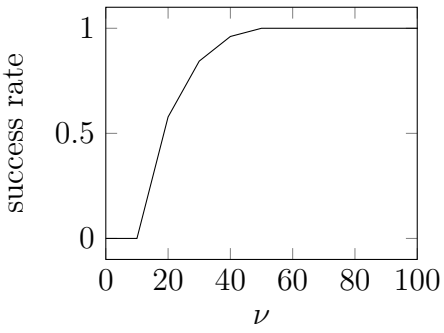
(a) Ψ_M (b) Ψ_P

Figure 6.11 Result of HOCPA on Kyber with $\nu = 1000$ and F_{abs} targeting $\hat{s}_0$ for both Ψ_M and Ψ_P. The correlation scores of incorrect hypotheses are in gray, and for the correct hypothesis in black.

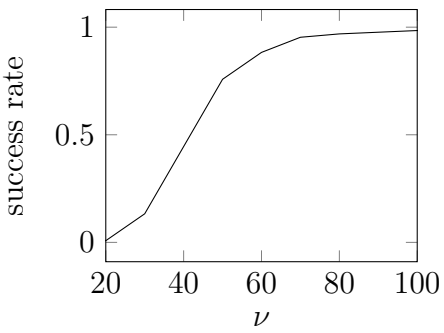


(a) HOCPAs on masked Ψ_m with F_{abs}

(b) HOCPAs on masked Ψ_p with F_{abs}



(c) CPAs on unmasked Ψ_m with HW_β



(d) CPAs on unmasked Ψ_p with HW_β

Figure 6.12 Success rates of the CPA and HOCPA attacks on Kyber. The success rate refers to ($^{\#\text{ correctly predicted }\hat{s}_{[i]}}/_{128}$). Retrieving $\pm\hat{s}_{[i]}$ is considered as success.

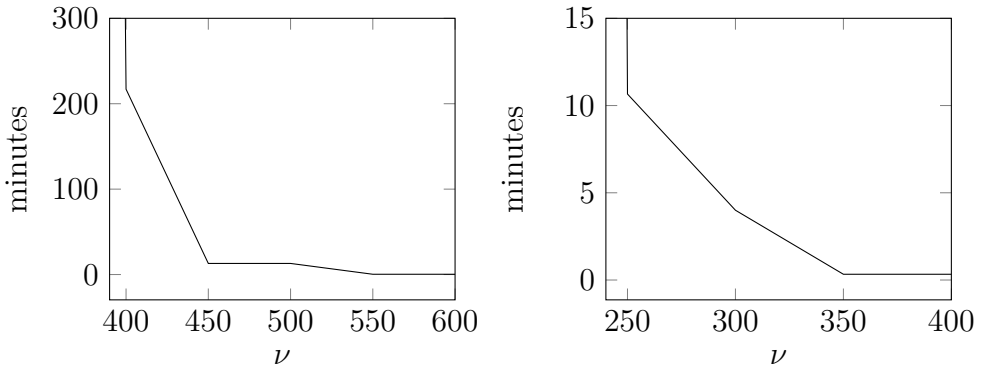
6.3.2.4. Evaluations

Let us start the evaluations by exemplary presenting the result of the individual attacks on $\hat{s}_{[0]}$ for both Ψ_M and Ψ_P in Figure 7.5. The correlation peaks for the correct hypotheses are observed in the corresponding time samples for the secret coefficient. Observe that the correlation for the correct hypotheses are around 0.3, which can be considered a major correlation for a higher-order attack. Needless to say, the correlation coefficient changes for different values of i. From a better perspective, Figure 6.12a and Figure 6.12b present the efficiency of our introduced prediction function F_{abs} in terms of the number of traces needed to succeed. Consistent with the simulation results, F_{abs} is very effective against arithmetic masking with central reduction. In particular, the attacks require 850 and 550 traces to fully recover the secret of the evaluated implementations. The reason why the attack on Ψ_M requires more traces to succeed is that the secret coefficients $\hat{s}_{[i]}$ for even values of i lead to lower correlation scores in general compared to the rest of the attack. While this observation can be micro-architecture and implementation specific, we did not concentrate on improving it as the overall attack still leads to a reasonably low number of traces. We should also remark that the aim of this study is not to compare Ψ_M and Ψ_P since both implementations employ central reduction; rather the goal is to show that the approach generalizes to central reduction techniques.

As a reference, we also included the success rate of a classical first-order CPA by the HW_β prediction function performed on the same but unprotected implementations in Figure 6.12c and Figure 6.12d. In order to keep the consistency, we used the same part of the power traces as those considered in HOCPAs. Distinctively, we used both lower- and higher-degree coefficients from the output of each $\hat{s}_{[i]} \cdot \hat{c}_{[i]}$ as the target function, namely $\mathsf{G}(\hat{s}_{[i]}, \hat{c}_{[i]}) = \hat{r}_{[i][0]} || \hat{r}_{[i][1]}$ (see Section 4.3). We should also note that F_{abs} is designed to work with a single coefficient, and we leave construction of a prediction function which takes multiple coefficients as the future work.

6.3.2.5. Combining with lattice attack

We also applied the lattice attack described in Section 4.4, iteratively executing the attack until a successful instance was found. The attacks were launched starting from the subset of NTT domain coefficients corresponding to the highest correlation scores. Each execution of the lattice attack takes 20 seconds, returning success in case of the included subset of coefficients was correctly retrieved by the preceding



(a) Ψ_M (b) Ψ_P

Figure 6.13 Time required for the lattice attacks to successfully retrieve the whole secret polynomial s in Kyber768, using the predictions for $\hat{s}_{[i]}$ which are obtained by HOCPAs.

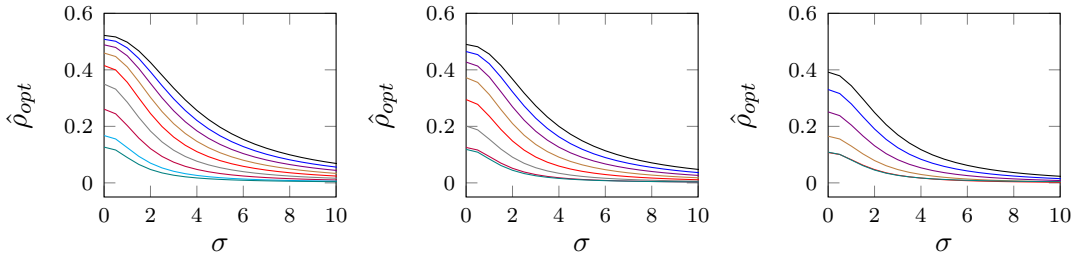
SCA attack. Additional details about this process are provided in Section 7.2.5.5. Figure 6.13 depicts the relationship between the number of traces and the time needed by the lattice attack to succeed in our experiments. In particular for Ψ_M, the attack succeeds after 643 trials (≈ 3.5 hours) with 400 traces while it succeeds with 250 traces for Ψ_P after 27 trials (≈ 10 minutes)[7]. Indeed, $\nu = 250$ is considered as a very small number to conduct a successful non-profiling higher-order SCA attack on a masked implementation. The lattice attack is implemented using fpylll library[8]. A more detailed discussion about the application of the lattice attack can be found in the next chapter.

6.4. Absolute Difference Combination Function

In the following analysis including Figure 6.14, Figure 6.15, Table 6.4, Table 6.5, and Table 6.6, for the sake of simplicity we took $\mu_0 = \mu_1 = 0$ regarding the noise terms in $\mathsf{L}(X^{\{0\}})$ and $\mathsf{L}(X^{\{1\}})$ (see Equation (3.45)). Note that, our results hold as long as $\mu_0 = \mu_1$. Estimation of the corresponding optimal prediction function – so-called $\hat{\mathsf{F}}^2_{\text{opt}}(i)$ – is performed with 100 K samples uniformly taken for $X^{\{0\}}$, $X^{\{1\}}$, and the noise taken from $\mathsf{N}(0, \sigma)$ for each i. Results that depend on the $\hat{\mathsf{F}}^2_{\text{opt}}$ for $q = 8380417$ are not presented due to the computational complexity of experiments. However,

[7] Except for $\nu = 250$ for Ψ_P and $\nu = 400$ for Ψ_M, the timing of lattice attacks are approximations.

[8] `https://pypi.org/project/fpylll/`



(a) $q = 257$ (b) $q = 769$ (c) $q = 3329$

Figure 6.14 Estimated optimal correlation for C_{abs} with respect to the noise standard deviation σ for different q, β and reduction algorithms. Estimations are performed with 1 million samples uniformly taken for $X^{\{0\}}$ and $X^{\{1\}}$.

- ★ Reduction to $[-q/2, q/2]$ for $q = 257$, $q = 769$ and $q = 3329$: $\beta = 16$ (black), $\beta = 15$ (blue), $\beta = 14$ (violet), $\beta = 13$ (brown), $\beta = 12$ (red), $\beta = 11$ (gray), $\beta = 10$ (purple), $\beta = 9$ (cyan)
- ★ Reduction to $[0, q)$ (teal)

they can be anticipated from the results for the other studied primes presented in this section and the ones corresponding to the mean-free product (Figure 6.3 and Table 6.3).

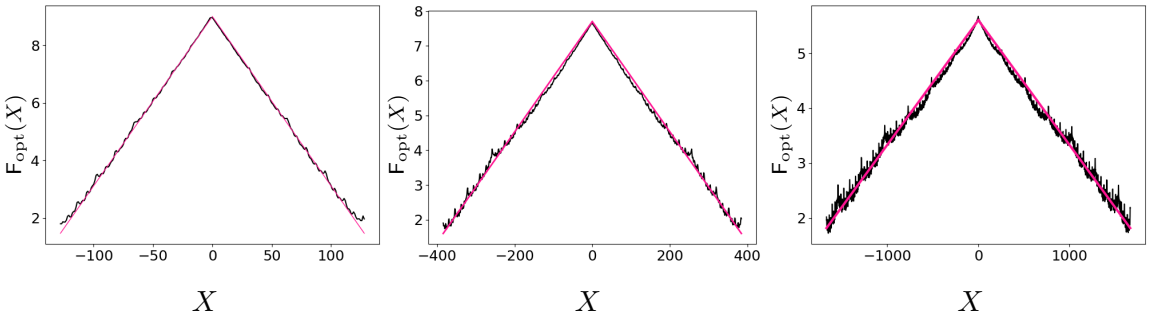
6.4.1. Optimal Correlation for C_{abs}

Demonstrated in Figure 6.14.

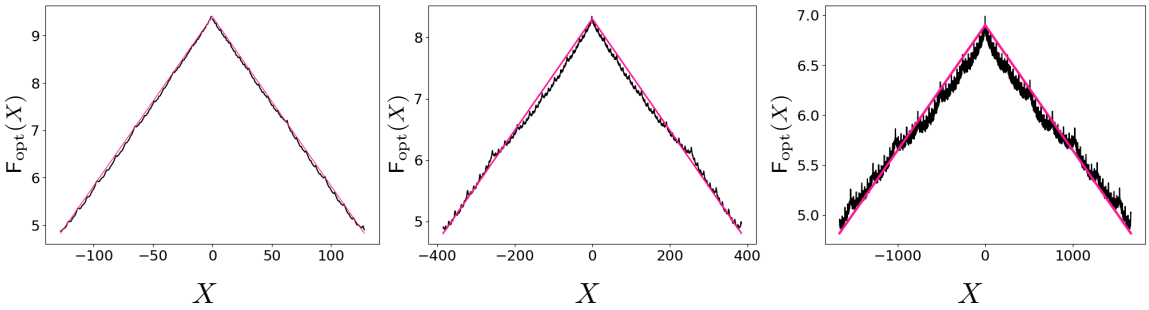
6.4.2. Accuracy of Absolute Value Prediction Function for C_{abs}

It can be seen in Figure 6.15 that $\hat{\mathsf{F}}^2_{\text{opt}}$ for C_{abs}, is an affine function of F_{abs} with some noise. Table 6.4, Table 6.5, and Table 6.6 present the estimations for $\hat{\rho}\left(\mathsf{F}_{\text{abs}}(X), \mathsf{F}^2_{\text{opt}}(X)\right)$ in this configuration. Observe that $|\hat{\rho}| > 0.99$ for the evaluated q and β couples mentioned in Section 6.1, allowing us to conclude that F_{abs} can be effectively used with C_{abs}.

Additionally, we provide the correlation results for $q = 8380417$ in Figure 6.16.



(a) $q = 257$, $\beta = 16$, $\sigma = 0$ (b) $q = 769$, $\beta = 16$, $\sigma = 0$ (c) $q = 3329$, $\beta = 16$, $\sigma = 0$



(d) $q = 257$, $\beta = 16$, $\sigma = 4$ (e) $q = 769$, $\beta = 16$, $\sigma = 4$ (f) $q = 3329$, $\beta = 16$, $\sigma = 4$

Figure 6.15 Visualization of optimal prediction function $\hat{\mathsf{F}}^2_{\mathrm{opt}}$ for the central reduction range $[-q/2, q/2]$ for different q and σ with respect to absolute difference combination function. $\hat{\mathsf{F}}^2_{\mathrm{opt}}(X)$ (black), $c'_0 \cdot |X| + c'_1$ (pink) for some c'_0 and c'_1.

Table 6.4 Estimated correlation between the absolute value and optimal prediction functions, $\hat{\rho}(\mathsf{F}_{\mathrm{abs}}(X), \hat{\mathsf{F}}^2_{\mathrm{opt}}(X))$ for $\mathsf{C}_{\mathrm{abs}}$, $q = 257$ and different β and σ. The estimations are performed with 1 million uniformly random samples for X while the reduction is central.

β \ σ	0	2	4	6	8	10
16	**−0.999**	**−0.999**	**−0.999**	**−0.999**	**−0.999**	**−0.999**
15	−0.999	−0.999	−0.999	−0.999	−0.999	−0.998
14	−0.998	−0.998	−0.998	−0.998	−0.998	−0.998
13	−0.998	−0.998	−0.997	−0.997	−0.996	−0.996
12	−0.996	−0.996	−0.994	−0.993	−0.993	−0.992
11	−0.992	−0.990	−0.986	−0.984	−0.983	−0.982
10	−0.975	−0.964	−0.957	−0.952	−0.949	−0.947
9	−0.864	−0.834	−0.818	−0.809	−0.801	−0.793

Table 6.5 Estimated correlation between the absolute value and optimal prediction functions, $\hat{\rho}(\mathsf{F}_{\text{abs}}(X), \hat{\mathsf{F}}^2_{\text{opt}}(X))$, for C_{abs}, $q = 769$ and different β and σ. The estimations are performed with 1 million uniformly random samples for X while the reduction is central.

β \ σ	0	2	4	6	8	10
16	**−0.998**	**−0.998**	**−0.998**	**−0.998**	**−0.998**	**−0.998**
15	−0.997	−0.998	−0.997	−0.997	−0.996	−0.996
14	−0.996	−0.996	−0.995	−0.994	−0.994	−0.994
13	−0.993	−0.992	−0.990	−0.989	−0.988	−0.988
12	−0.985	−0.979	−0.974	−0.972	−0.971	−0.970
11	−0.941	−0.923	−0.912	−0.908	−0.905	−0.902
10	−0.673	−0.645	−0.632	−0.626	−0.620	−0.614

Table 6.6 Estimated correlation between the absolute value and optimal prediction functions, $\hat{\rho}(\mathsf{F}_{\text{abs}}(X), \hat{\mathsf{F}}^2_{\text{opt}}(X))$ for C_{abs}, $q = 3329$ and different β and σ. The estimations are performed with 1 million uniformly random samples for X while the reduction is central.

β \ σ	0	2	4	6	8	10
16	**−0.996**	**−0.995**	**−0.993**	**−0.993**	**−0.992**	**−0.992**
15	−0.992	−0.989	−0.986	−0.985	−0.984	−0.983
14	−0.978	−0.971	−0.965	−0.962	−0.961	−0.959
13	−0.913	−0.893	−0.882	−0.877	−0.872	−0.869
12	−0.562	−0.551	−0.544	−0.539	−0.534	−0.529

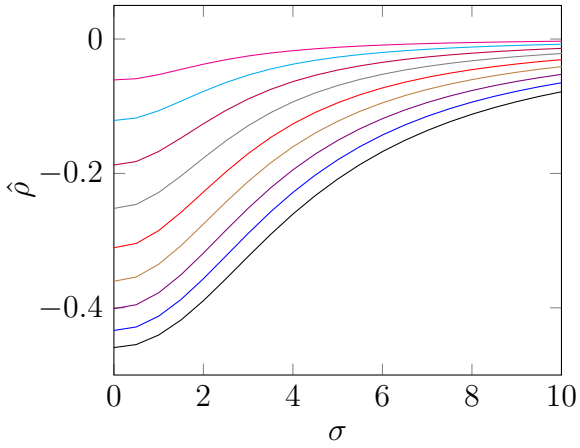


Figure 6.16 Estimations of $\hat{\rho}(\mathsf{F}_{\mathrm{abs}}(X), \mathsf{C}_{\mathrm{abs}}(X^{\{0\}}, X^{\{1\}}))$ with respect to the noise standard deviation σ for $q = 8380417$ and different β. The estimations are performed with 1 million samples uniformly taken for $X^{\{0\}}$ and $X^{\{1\}}$.

★ Reduction to $[-q/2, q/2]$ for $q = 8380417$: $\beta = 32$ (black), $\beta = 31$ (blue), $\beta = 30$ (violet), $\beta = 29$ (brown), $\beta = 28$ (red), $\beta = 27$ (gray), $\beta = 26$ (purple), $\beta = 25$ (cyan), $\beta = 24$ (magenta).

7. HIGHER-ORDER ATTACKS

This chapter is based on Section 3 an Section 5.1 of the publication (Tosun et al., 2025).

We develop efficient higher-order HOCPA attacks in which the attacker must compute a function known as the optimal prediction function. We revisit the definition of the optimal prediction function (see Equation (3.58)) and introduce a recursive method for computing it efficiently. Our approach is particularly useful when a closed-form formula is unavailable, as in LBC. Then, we introduce **sin** and **cos** prediction functions, which prove optimal for HOCPA attacks against second and higher-order masking. We validate our methods through simulations and real-device experiments on open-source masked implementations of Dilithium and Kyber on an Arm Cortex-M4. On the real device, we achieve full secret-key recovery using only **700** and **2400** traces for second and third-order masked implementations of Dilithium, and **2200** and **14500** traces for second and third-order masked implementations of Kyber, respectively.

7.1. New Prediction Functions for HOCPA

7.1.1. Recursive Computation of OPF

In this section, we present an efficient approach for the computation of the optimal prediction function $\mathsf{F}^d_{\text{opt}}(x)$ presented in Equation (3.58) based on recursion. Consider masking with d shares.

In case Equation (3.58) does not have an explicit formula, as in the case of arithmetic

Algorithm 13 Computation of $\mathsf{F}^d_{\text{opt}}$ as a look-up table for masking with d shares.

Input: Modulus q
Input: Predicted Leakage Function L'
Input: Number of shares d
Output: Look-up table $\mathtt{F}$ representing the function $\mathsf{F}^d_{\text{opt}}$

for $x = 0$ to $q-1$ **do**
  $\mathtt{F}_{[x]} = 0$
  for all $(x_0, x_1, \ldots, x_{d-2}) \in \{0, \ldots, q-1\}^{d-1}$ **do** ▷ or $\in \{-q/2, \ldots, q/2\}^{d-1}$
    $x_{d-1} = \left(x - \sum_{i=0}^{d-2} x_i\right) \pmod q$ ▷ $\bmod^+ q$ or $\bmod^\pm q$
    $p = 1$
    for $i = 0$ to $d-1$ **do**
      $p = p \cdot \mathsf{L}'(x_i)$
    end for
    $\mathtt{F}_{[x]} = \mathtt{F}_{[x]} + p$
  end for
end for
return $\mathtt{F}$

masking, $\mathsf{F}^d_{\text{opt}}(x)$ can be calculated using a computer in $O(q^d)$. The computation procedure is presented in Algorithm 13[1]. Needless to say, this can be very expensive for higher-order masking. For instance, with $d = 4$ and $\log q = 23$ (*e.g.* third-order masking for Dilithium), $q^d \approx 2^{92}$. We present a better solution by a recursive approach. For $d \geq 2$, we apply the following formula:

$$\mathsf{F}^d_{\text{opt}}(x) = \mathsf{E}\Big[\mathsf{C}\Big(f_{opt}^{\lceil d/2 \rceil}(T), f_{opt}^{\lfloor d/2 \rfloor}(X - T)\Big)\Big| X = x\Big] \tag{7.1}$$

where random variables X and T take values in the discrete space $\mathtt{X}$, *e.g.* $\mathbb{Z}_q$, and T is uniformly distributed. As the base case of the recursion, $\mathsf{F}^1_{\text{opt}}(x)$ is defined to be the predicted device leakage function, $\mathsf{F}^1_{\text{opt}}(x) = \mathsf{L}'(x)$. This approach takes $O(\log dq^2)$ computational time with dynamic programming. The computation of the optimal prediction function becomes feasible for large d with our recursive approach. Notice that $\log dq^2 \approx 2^{47}$ for the running example.

Proof: We begin by reformulating the definition of the optimal prediction function $\mathsf{F}^d_{\text{opt}}(x)$ for the mean-free product:

$$\mathsf{F}^d_{\text{opt}}(x) = \mathsf{E}\left[\prod_{i=0}^{d-1} \mathsf{L}'(X^{\{i\}})\middle| X = x\right] + \Gamma \tag{7.2}$$

[1] Additions by constants and divisions by constants are omitted. Therefore, both Algorithm 13 and Algorithm 14 compute $\mathsf{F}^d_{\text{opt}}(x) \cdot c_0 + c_1$ for some constants c_0 and c_1 which are the same for all x.

where Γ accumulates the constant terms arising from $\mathsf{E}[\mathsf{L}'(X^{\{i\}})]$. We omit this term, as it is independent of x and does not influence the correlation.

Next, we prove Equation (7.1) using induction. The base case $\mathsf{F}^1_{\text{opt}}(x)$ corresponds to a prediction function for an unprotected target, which is naturally $\mathsf{L}'(x)$ itself as in the first-order CPA. For simplicity, consider even $d \geq 2$ and assume that $\mathsf{F}^{d/2}_{\text{opt}}(x)$ is correct. Plugging in Equation (7.1):

(7.3)
$$\mathsf{F}^d_{\text{opt}}(x) = \mathsf{E}\left[\mathsf{C}\left(\mathsf{E}\left[\prod_{i=0}^{d/2-1} \mathsf{L}'(X^{\{i\}}) \middle| X_0 = T\right], \mathsf{E}\left[\prod_{i=d/2}^{d-1} \mathsf{L}'(X^{\{i\}}) \middle| X_1 = X - T\right]\right) \middle| X = x\right]$$

where $X = \sum_{i=0}^{d/2-1} X^{\{i\}}$, $X_0 = \sum_{i=0}^{d/2-1} X^{\{i\}}$ and $X_1 = \sum_{i=d/2}^{d-1} X^{\{i\}}$. $X^{\{0:d-1\}}$ are uniformly random over $\mathtt{X}$. By re-writing the inner expectations as the sum of products,

(7.4)
$$\begin{aligned}\mathsf{F}^d_{\text{opt}}(x) = \mathsf{E}\Bigg[&\frac{1}{q^{d/2-1}}\Bigg(\sum_{x^{\{0\}}\in\mathtt{X}}\sum_{x^{\{1\}}\in\mathtt{X}}\cdots\sum_{x^{\{d/2-2\}}\in\mathtt{X}}\prod_{i=0}^{d/2-1}\mathsf{L}'(x^{\{i\}})\Bigg)\cdot \\ &\frac{1}{q^{d/2-1}}\Bigg(\sum_{x^{\{d/2\}}\in\mathtt{X}}\sum_{x^{\{d/2+1\}}\in\mathtt{X}}\cdots\sum_{x^{\{d-2\}}\in\mathtt{X}}\prod_{i=d/2}^{d-1}\mathsf{L}'(x^{\{i\}})\Bigg)\Bigg| X = x\Bigg]\end{aligned}$$

where $x^{\{d/2-1\}} = T - \sum_{i=0}^{d/2-2} x^{\{i\}}$ and $x^{\{d-1\}} = X - T - \sum_{i=d/2}^{d-2} x^{\{i\}} = X - \sum_{i=0}^{d-2} x^{\{i\}}$. Combining the products, we have:

(7.5)
$$\mathsf{F}^d_{\text{opt}}(x) = \frac{1}{q^{d-2}}\mathsf{E}\left[\Bigg(\sum_{x^{\{0\}}\in\mathtt{X}}\sum_{x^{\{1\}}\in\mathtt{X}}\cdots\sum_{x^{\{d/2-2\}}\in\mathtt{X}}\sum_{x^{\{d/2\}}\in\mathtt{X}}\cdots\sum_{x^{\{d-2\}}\in\mathtt{X}}\prod_{i=0}^{d-1}\mathsf{L}'(x^{\{i\}})\Bigg)\middle| X = x\right]$$

Taking the expectation over T completes the missing summation term over $x^{\{d-2\}}$:

$$\mathsf{F}^d_{\text{opt}}(x) = \frac{1}{q^{d-1}}\sum_{x^{\{0\}}\in\mathtt{X}}\sum_{x^{\{1\}}\in\mathtt{X}}\cdots\sum_{x^{\{d-2\}}\in\mathtt{X}}\left(\left(\prod_{i=0}^{d-2}\mathsf{L}'(x^{\{i\}})\right)\cdot\mathsf{L}'\left(x - \sum_{i=0}^{d-2}x^{\{i\}}\right)\right) \tag{7.6}$$

which is equivalent to the definition of the optimal prediction function given in Equation (3.58). ■

Table 7.1 Estimations of the optimal correlation, $\hat{\rho}(\mathsf{C}(\{\mathsf{HW}_\beta(X^{\{i\}})\}_{i=0}^{d-1}), \mathsf{F}_{\text{opt}}^d(X))$, for different q, β, d, σ and modular reduction strategies (*i.e.* central or not). The estimations are performed with 1 million uniformly random samples for $X^{\{0:d-1\}}$ drawn from the discrete space $\mathtt{X}$. The rows are ordered based on correlation magnitudes. Estimated values are rounded to nearest.

(a) $q = 3329$, $\beta = 16$

		σ					
$\mathtt{X}$	d	0	2	4	6	8	10
$[-q/2, q/2]$	1	1.00	0.85	0.63	0.47	0.37	0.31
$[-q/2, q/2]$	2	0.41	0.30	0.16	0.09	0.06	0.04
$[-q/2, q/2]$	3	0.21	0.13	0.05	0.02	0.01	0.01
$[-q/2, q/2]$	4	0.11	0.06	0.02	0.01	0.00	0.00
$[-q/2, q/2]$	5	0.06	0.03	0.01	0.00	0.00	0.00
$[0, q)$	1	1.00	0.63	0.38	0.26	0.20	0.16
$[0, q)$	2	0.17	0.07	0.03	0.01	0.01	0.01
$[0, q)$	3	0.03	0.01	0.00	0.00	0.00	0.00
$[0, q)$	4	0.01	0.00	0.00	0.00	0.00	0.00
$[0, q)$	5	0.00	0.00	0.00	0.00	0.00	0.00

(a) $q = 8380417$, $\beta = 32$

		σ					
$\mathtt{X}$	d	0	2	4	6	8	10
$[-q/2, q/2]$	1	1.00	0.94	0.81	0.68	0.57	0.48
$[-q/2, q/2]$	2	0.47	0.42	0.31	0.22	0.15	0.11
$[-q/2, q/2]$	3	0.27	0.22	0.14	0.08	0.05	0.03
$[-q/2, q/2]$	4	0.16	0.12	0.07	0.03	0.02	0.01
$[-q/2, q/2]$	5	0.09	0.07	0.03	0.01	0.01	0.00
$[0, q)$	1	1.00	0.77	0.51	0.37	0.29	0.23
$[0, q)$	2	0.12	0.07	0.03	0.01	0.01	0.00
$[0, q)$	3	0.02	0.01	0.00	0.00	0.00	0.00
$[0, q)$	4	0.00	0.00	0.00	0.00	0.00	0.00
$[0, q)$	5	0.00	0.00	0.00	0.00	0.00	0.00

Algorithm 14 Computation of $\mathsf{F}^d_{\text{opt}}$ as a look-up table using the recursive formula Equation (7.1).

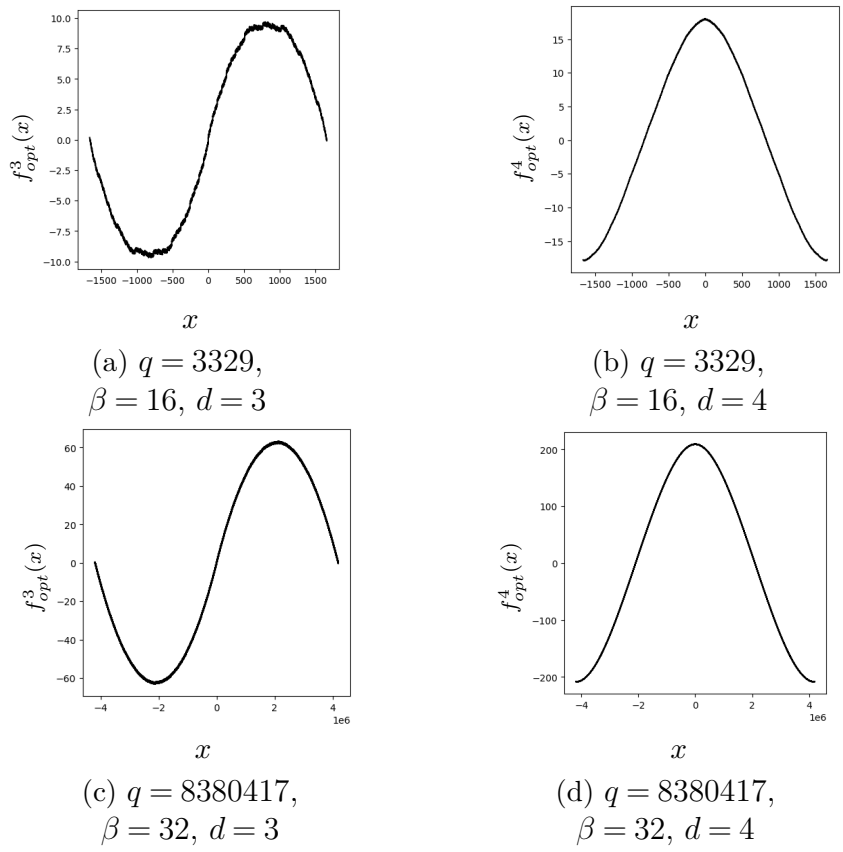
Input: Modulus q
Input: Predicted Leakage Function L'
Input: Number of shares d
Output: Look-up table $\mathtt{F}^d$ representing the function $\mathsf{F}^d_{\text{opt}}$

if $d = 1$ **then**
  for $x = 0$ to $q-1$ **do**
    $\mathtt{F}^1_{[x]} = \mathsf{L}'(x)$
  end for
  return $\mathtt{F}^1$
end if
$\mathtt{F}^{\lceil d \rceil}$ =Algorithm 14(q, L', $\lceil d \rceil$)
$\mathtt{F}^{\lfloor d \rfloor}$ =Algorithm 14(q, L', $\lfloor d \rfloor$) ▷ Recursive computations not repeated for the same d.
for $x = 0$ to $q-1$ **do**
  $\mathtt{F}^2_{opt[x]} = 0$
  for $x_0 = 0$ to $q-1$ **do** ▷ or $x_0 = -q/2$ to $q/2$
    $x_1 = x - x_0 \pmod q$, ▷ $\text{mod}^+ q$ or $\text{mod}^\pm q$
    $\mathtt{F}^d_{[x]} = \mathtt{F}^d_{[x]} + \mathtt{F}^{\lceil d \rceil}_{[x_0]} \cdot \mathtt{F}^{\lfloor d \rfloor}_{[x_1]}$
  end for
end for
return $\mathtt{F}^d$

7.1.2. Optimal Correlation

Using the presented recursive computation approach, we computed a set of optimal prediction functions $\mathsf{F}^d_{\text{opt}}$ for the SCA attack targeting NTT multiplication, as outlined in Section 3.5.3. Recall that the target function is modular multiplication, and the target intermediate X is masked as $X = \sum_{i=0}^{d-1} X^{\{i\}} \pmod q$. Specifically, for this scenario, we computed $\mathsf{F}^d_{\text{opt}}(x)$ as a look-up table for all $x \in \mathbb{Z}_q$. The computation procedure is presented in Algorithm 14. We consider Hamming weight leakage function HW_β. The computations of look-up table based $\mathsf{F}^d_{\text{opt}}(x)$ were carried out for $2 \leq d \leq 5$, $\forall q \in \{3329, 8380417\}$ and under both central and non-central modular reduction. Let β denote the machine word size in bits, which is typically 16 or 32 in software implementations. We consider $\beta = 16$ for $q = 3329$ and $\beta = 32$ for $q = 8380417$, consistent with the configurations used in Kyber and Dilithium implementations (Bronchain & Cassiers, 2022; Coron et al., 2024; Heinz et al., 2022; Kannwischer et al., 2019). Then, we computed the optimal correlation using these functions, demonstrated in Table 7.1. We studied both the noiseless case, where $\sigma = 0$, and scenarios with increasing noise levels, up to $\sigma = 10$. Observe that employing a central reduction range increases the correlation confirming the results

presented in Chapter 6 and extending them to higher-order. Interestingly, it significantly decreases the impact of masking order. For instance, for $q = 83801417$, $d = 4$ with central reduction leads to higher correlation amounts compared to $d = 2$ with the unsigned reduction.



(a) $q = 3329$, $\beta = 16$, $d = 3$

(b) $q = 3329$, $\beta = 16$, $d = 4$

(c) $q = 8380417$, $\beta = 32$, $d = 3$

(d) $q = 8380417$, $\beta = 32$, $d = 4$

Figure 7.1 Visualization of optimal prediction function $\mathsf{F}^d_{\text{opt}}$ for the central reduction range $[-q/2, q/2]$ and different q, d, β.

7.1.3. Explicit Formulas

When the reduction is central, *i.e.* the secret shares $X^{\{0:d-1\}} \in \mathtt{X}$ where $\mathtt{X} = [-q/2, q/2]$, the device leakage corresponds to the Hamming weight, and there is a sufficient margin between β and $\log q$, which is the case for $\{q = 3329, \beta = 16\}$ and $\{q = 8380417, \beta = 32\}$, it is possible to approximate $\mathsf{F}^d_{\text{opt}}$ with an explicit formula, denoted by $\mathsf{F}^{*d}_{\text{opt}}$. It is shown in Chapter 6 that $\mathsf{F}^{*2}_{\text{opt}}(x) = \mathsf{F}_{\text{abs}} = |x|$. In this section, we present explicit formulas for higher-orders, $d > 2$, and show that these formulas are optimal. In fact, these formulas are derived from Equation (7.1). Figure 7.1 visualizes the $\mathsf{F}^d_{\text{opt}}$ for $d = 3$ and $d = 4$, providing an intuition for finding closed formulas. Table 8.1 presents the closed-form formulas for the estimations $\mathsf{F}^{*d}_{\text{opt}}$. A

common technique to compute the accuracy of a prediction function is to compute its correlation to the optimal prediction function (Prouff et al., 2009). Observe that presented formulas are highly accurate for the moduli employed by Kyber and Dilithium.

Table 7.2 Estimated correlation between the optimal prediction function and its proposed approximation, $\hat{\rho}(\mathsf{F}^{*d}_{\text{opt}}(X), \mathsf{F}^{d}_{\text{opt}}(X))$, for different q, β, d when the modular reduction is central. The estimations are performed with 1 million uniformly random samples for X.

		q	β	q	β
d	$f^{*d}_{opt}(x)$	3329	16	8380417	32
3	$\sin(2\pi x/q)$	0.997		0.999	
4	$\cos(2\pi x/q)$	0.999		0.999	
5	$-\sin(2\pi x/q)$	0.999		0.999	
6	$-\cos(2\pi x/q)$	0.999		0.999	
7	$\sin(2\pi x/q)$	0.999		0.999	
8	$\cos(2\pi x/q)$	0.999		0.999	

7.2. Results

In this section, we present experimental results for the side-channel attacks introduced in Section 7.1. To support reproducibility, the source code for trace acquisition, attack scripts, and the simulations presented in the previous section are made publicly available in `https://github.com/toluntosun21/HighOrderNonProfiledSCALBC`.

7.2.1. Attack Setup

For the analysis and attack, we utilized a modified version of the open-source side-channel analysis library scared. In particular, we ported scared to GPU using cupy[2],

[2]https://pypi.org/project/cupy/

and made it publicly accessible at `https://github.com/toluntosun21/scaredcu`.

We run the experiments on a supercomputer with NVIDIA RTX4090 GPU. For the application of lattice attacks, we use the fpylll library.

7.2.2. Attack Results

We perform the side-channel attacks presented in Section 7.1. We begin with simulated traces, where power samples are intentionally crafted to reflect Hamming weight leakage. Subsequently, we conduct real-device experiments using publicly available implementations of Kyber and Dilithium in an embedded setting.

Table 7.3 Summary of prediction functions employed in our (HO)CPA attacks.

Reduction Range	d	**Prediction Function**
$[-q/2, q/2]$	1	Hamming weight
$[-q/2, q/2]$	2	*abs* (**this work**, Chapter 6)
$[-q/2, q/2]$	3	*sin* (**this work**)
$[-q/2, q/2]$	4	*cos* (**this work**)
$[0, q)$	1	Hamming weight
$[0, q)$	2	OPF (Prouff et al. (2009))
$[0, q)$	3	Recursive OPF (**this work**)
$[0, q)$	4	Recursive OPF (**this work**)

7.2.3. (HO)CPA Details

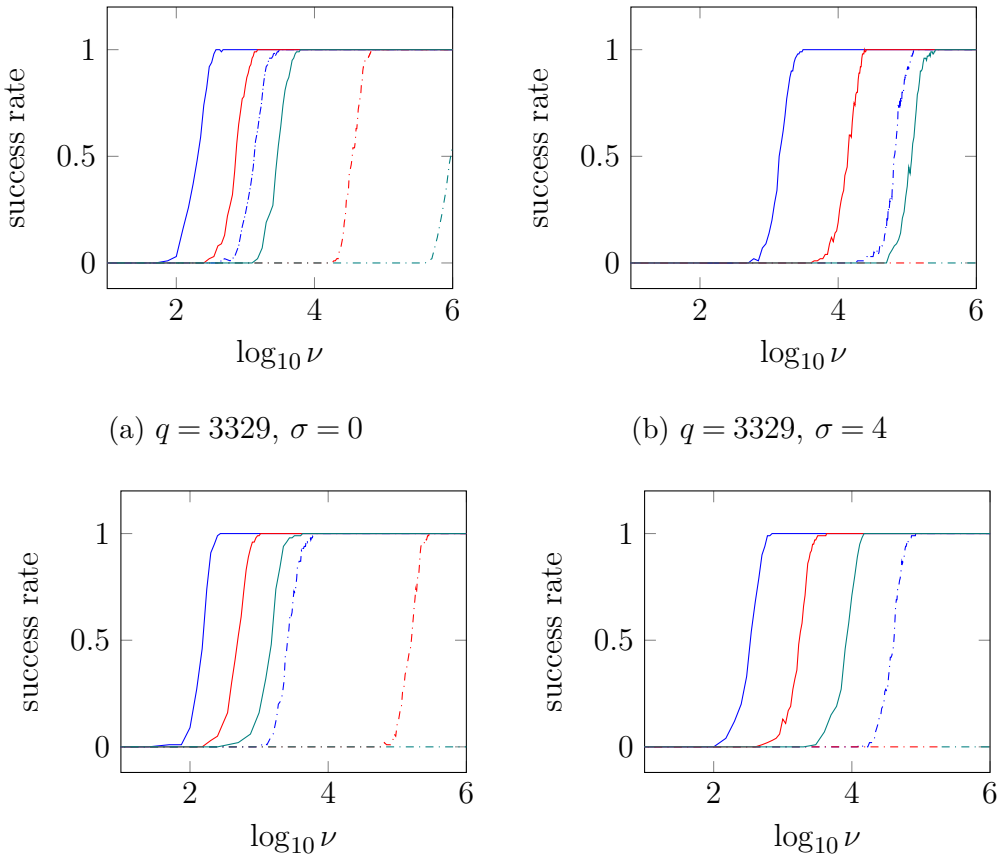
We perform (HO)CPA attacks for $d \in \{1, 2, 3, 4\}$. For attacking the base multiplication with central reduction, we employ the prediction functions with explicit formulas discussed earlier. Specifically, for $d = 2$, we use the absolute value prediction function from Chapter 6. For $d = 3$ and $d = 4$, we apply the *sin* and *cos* prediction functions introduced in Section 7.1.3. In the case of the non-central reduction, we follow the methodology described in Section 7.1.1 to compute the prediction functions computationally and recursively. Note that in certain scenarios (*e.g.* $d = 4$, $q = 8380417$), computing the prediction function and performing the

HOCPA would not be computationally feasible possible without such optimization. For $d = 1$, we employ the Hamming weight as the prediction function, which is a common practice in the literature, especially for attacking software implementations. Table 7.3 summarizes the prediction functions used in our (HO)CPA attacks. For all masked scenarios, we employed the mean-free product combination C. The target intermediate is chosen as the higher-degree coefficient $\hat{r}_{[i][1]}$ from the output of base multiplication for attacking Kyber's incomplete NTT for $d > 1$. For $d = 1$, we employ $\hat{r}_{[i][0]}||\hat{r}_{[i][1]}$. For both Kyber and Dilithium, and when the reduction range is central, we reduce the hypothesis space by half by leveraging the observation that additive inverse of the actual secret also produce peaks in the HOCPA results, as discussed in Chapter 4. This optimization has a negligible impact on attack accuracy. Consequently, we tested $q/2$ hypotheses for Dilithium and $q^2/2$ hypotheses for Kyber due to incomplete NTT arithmetic.

7.2.4. HOCPA Attacks through Simulations

The simulated traces for the base multiplication operation were generated using the routine described in Algorithm 11. We considered various values for the noise standard deviation, $\sigma \in \{0, 4\}$, and the number of shares, $d \in \{2, 3, 4\}$. As in Section 6.3, we set $m = 100$ and $\mu = 0$ for all simulations, and evaluated both central (using $f_2 = 1$ in Algorithm 11) and unsigned reduction ($f_2 = 0$). The leakage function was set to Hamming weight, $\mathsf{L} = \mathsf{HW}_\beta$. For $q = 3329$, we simulated incomplete NTT arithmetic (by setting $f_1 = 1$ in Algorithm 11), while for $q = 8380417$, we simulated complete NTT arithmetic ($f_1 = 0$).

We performed a series of HOCPA attacks for $d \in \{2, 3, 4\}$ for both the central and non-central reduction range. Figure 7.2 demonstrates the results of the HOCPA simulations, in terms of the success rate with respect to ν. The success rate is defined as the number of correctly predicted elements of the secret vector, $\pm\mathbf{s}_{[i]}$, divided by m. The results confirm the findings of Chapter 6 and extend them to higher orders, *i.e.* $d > 2$, the central reduction is significantly easier to attack. Our findings further suggest that the hardness gap between central and non-central reduction strategies widens at higher orders. Observe that attacking Dilithium's modulus for $d = 4$ under central reduction requires fewer traces than the case for $d = 2$ under the unsigned reduction range. A similar trend is observed for $q = 3329$.



(a) $q = 3329$, $\sigma = 0$

(b) $q = 3329$, $\sigma = 4$

(c) $q = 8380417$, $\sigma = 0$

(d) $q = 8380417$, $\sigma = 4$

Figure 7.2 Success rates of HOCPA attacks on simulated traces generated with different q, d, σ and reduction ranges with respect to ν. For each point in the curves, 100 experiments have been performed with random data.

- ★ Reduction ranges: $[-q/2, q/2]$ (solid), $[0, q)$ (dashdotted).
- ★ Number of shares: $d = 2$ (blue), $d = 3$ (red), $d = 4$ (green).

7.2.5. HOCPA Attacks on Real Traces

7.2.5.1. Target implementations

We targeted the Kyber implementation of Bronchain & Cassiers (2022)[3], which is specific to ARM Cortex-M4 and supports masking at arbitrary orders, as we did in Chapter 6. This implementation inherits the polynomial arithmetic routines from the pqm4 library Kannwischer et al. (2019), which offers the state-of-the-art implementations of PQC algorithms on the Cortex-M4. Those parts are highly optimized and mostly written in assembly. For example, the degree-1 polynomial multiplications in Equation (3.35) and Equation (3.36) (during the base multiplication in incomplete NTT domain) are implemented using the `SMUAD(X)` instruction explained in Section 3.3.2. The function that implements the base multiplication in incomplete NTT domain is named `basemul_asm`, which is executed for each share. The modular reduction is performed using Montgomery reduction, which takes 2 clock cycles and outputs in the central range $(-q, q)$. For Dilithium, we targeted the open-source implementation of Coron et al. (2024)[4]. The polynomial arithmetic in this implementation is written in C language and compiled for the Cortex-M4, in contrast to the target Kyber implementation. The function that performs the base multiplication is named `poly_pointwise_montgomery`. As the name suggests, it employs Montgomery reductions similar to the to target Kyber implementation. However, these are more costly due to Dilithium's parameters requiring 32-bit arithmetic. For both Kyber and Dilithium, we focus on the medium security levels, *i.e.* Kyber768 and Dilithium3. It is important to note that our methods are applicable to all security levels. Additionally, our analysis focuses on attacking a single secret polynomial s, although the secret key in both Kyber and Dilithium contains a vector of polynomials. The methodology we present can be applied iteratively to target each polynomial in the vector.

[3] `https://github.com/uclcrypto/pqm4_masked/` commit hash: **5fe90ba**

[4] `https://github.com/fragerar/tches24_masked_Dilithium/tree/master` commit hash: **5b5fd32**

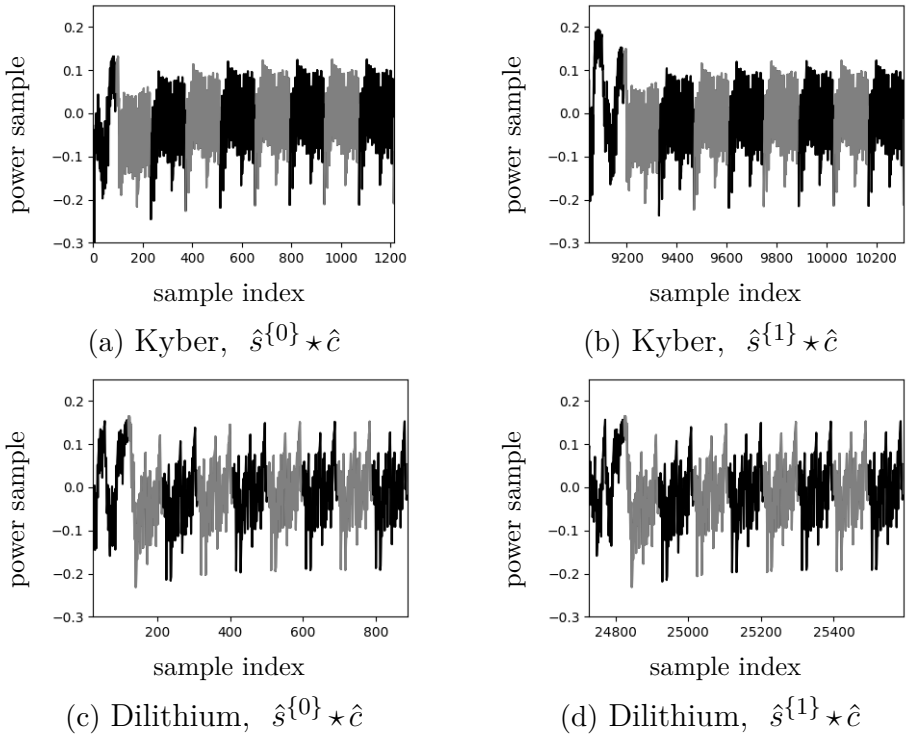
7.2.5.2. Trace collection

We used the same trace collection setup introduced in Section 6.3.2.2, with the CW1200 capturing traces and an STM32F303 microcontroller executing the victim implementations. A trigger signal was provided to the power-collection module to mark the beginning of the base multiplication function corresponding to the first share for both Kyber and Dilithium. The power samples are sampled at a rate of four samples at each clock cycle. Exceptionally, we decreased the sampling rate to one sample per clock cycle for Dilithium and $d = 4$. This limitation arises from the insufficient buffer size of the employed oscilloscope CW1200, which could not accommodate all samples from the beginning of the base multiplication for the first share to the end of the last share when capturing four samples per clock cycle. To maintain consistency in the presented results, we implicitly multiply the sample indices by four in the affected case.

7.2.5.3. Point-of-Interest detection

Figure 7.3 demonstrates the mean traces collected from the victim implementations for both Kyber and Dilithium. Observe that the iterations of the base multiplications are clearly distinguishable as the same pattern repeats consistently throughout the loop. Each iteration corresponds to 140 samples for Kyber and 96 samples for Dilithium. Note that the target implementation employs loop-unrolling; thus 140 samples actually cover two iterations of the base multiplication resulting in 64 total iterations instead of the expected 128. The pattern remains identical across all shares, thereby making the combination of leakage from different shares straightforward. As in Chapter 6, we used a constant offset to combine the leakage from different shares. The offset is equal to the total number of power samples that the base multiplication takes, along with additional samples due to loop overhead. We determined this offset through pattern matching. In particular, to attack the secret coefficient $\hat{s}_{[i]}$ in Dilithium, we focused on the set of power samples with indexes $\{150 + 96i + 24420j + \delta\}_{0 \leq j < d}$ as input to the combination function C for $0 \leq \delta < 96$. To further refine the focus of the attack to a specific value of δ, we performed the HOCPA attack for all possible values of δ and identified the one that maximized the product of the highest correlation scores across all hypotheses and all i. Formally, we computed $\delta' = \mathsf{argmax}_\delta \Pi_{i=0}^{n-1} \Psi_{[\delta]}^i$ where $\Psi_{[\delta]}^i = \mathsf{max}_\kappa(\mathsf{Score}_\delta(\mathtt{H}_{[\kappa]}^i))$, $\mathtt{H}^i$ denotes the set of hypotheses for $\hat{s}_{[i]}$ and Score_δ outputs the correlation score for the given hypoth-

esis and δ. Recall that each $\hat{s}_{[i]}$ is attacked independently. δ' identifies the point of interest (PoI), most strongly associated power sample with the targeted operation, namely $\hat{s}_{[i]} \cdot \hat{c}_{[i]}$. Figure 7.4 illustrates the process. The same process is applied to Kyber by adjusting the power sample offsets corresponding to each iteration and the offset between shares. Notice that we leverage the fact that δ' must remain the same across attacks on each $\hat{s}_{[i]}$. The Point-of-Interest (PoI) detection mechanism explained in this section does not require profiling.



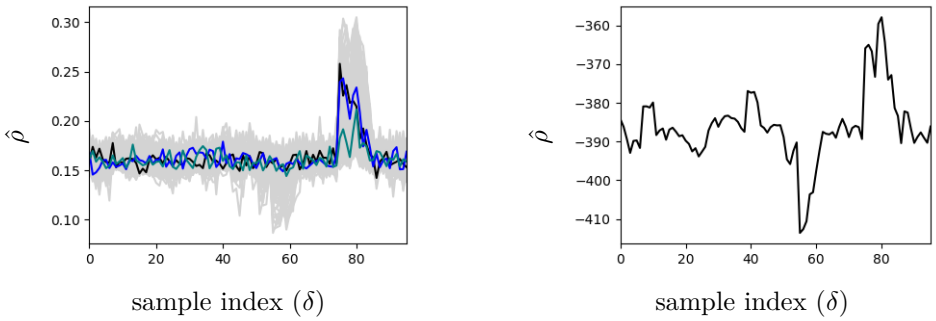
(a) Kyber, $\hat{s}^{\{0\}} \star \hat{c}$ (b) Kyber, $\hat{s}^{\{1\}} \star \hat{c}$

(c) Dilithium, $\hat{s}^{\{0\}} \star \hat{c}$ (d) Dilithium, $\hat{s}^{\{1\}} \star \hat{c}$

Figure 7.3 The mean power trace associated with the execution of the base multiplication functions of the attacked implementations of Kyber and Dilithium. The plots demonstrates the first eight iterations of main loops of the base multiplications, focusing on the first two shares (note that Sub-figures b and d are not relevant for implementations with $d = 1$). The observed pattern repeats consistently across all iterations and shares.

7.2.5.4. Evaluation of (HO)CPA attacks

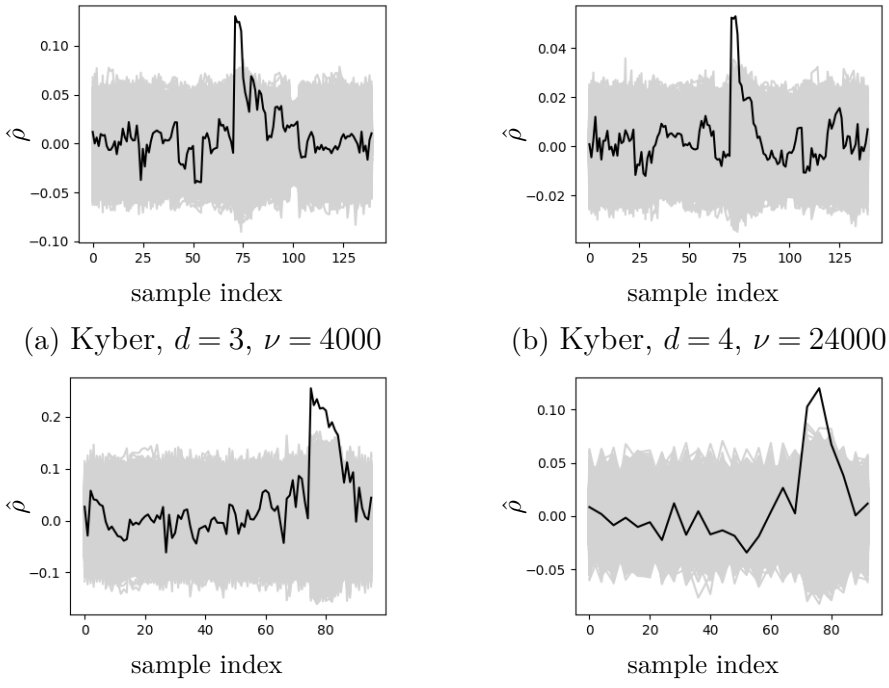
Figure 7.5 presents the HOCPA results against $\hat{s}_0$ as an example for $d = 3$ and $d = 4$. Observe that the actual secret is clearly distinguishable for each case. The correlation amounts comply with our estimations presented in Table 7.1. For Dilithium, the peaks are observed for significantly lower number of traces compared to Kyber. Figure 7.6 presents the success rates of attacks with respect to the available number of traces, measured as the ratio of successfully retrieved NTT domain secret



(a) $\Psi^0_{[\delta]}$, $\Psi^1_{[\delta]}$, $\Psi^2_{[\delta]}$ in black, blue and green, $\Psi^i_{[\delta]}$ for $3 \leq i < n = 256$ in gray

(b) $\sum_{i=0}^{n-1} \log \Psi^i_\delta$

Figure 7.4 Illustration of PoI identification for Dilithium for $d = 3$ and $\nu = 1000$.



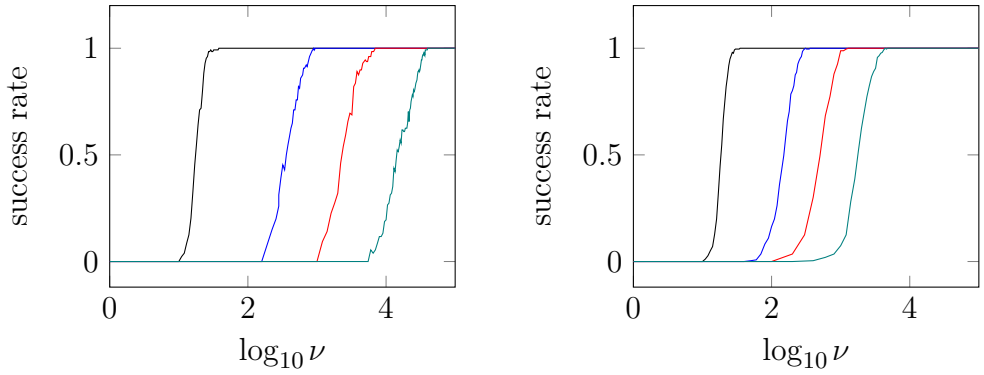
(a) Kyber, $d = 3$, $\nu = 4000$

(b) Kyber, $d = 4$, $\nu = 24000$

(c) Dilithium, $d = 3$, $\nu = 1000$

(d) Dilithium, $d = 4$, $\nu = 4000$

Figure 7.5 Results of HOCPA attacks on the victim Kyber and Dilithium implementations targeting $\hat{s}_{[0]}$ for the given d with respect to ν. The correlation scores corresponding to incorrect hypotheses are shown in gray, score for the correct hypothesis is shown in black.



(a) Kyber (b) Dilithium

Figure 7.6 Success rates of (HO)CPA attacks on (masked) Kyber and Dilithium implementations by Bronchain & Cassiers (2022) and Coron et al. (2024), respectively. Attacks are performed against different number of shares in the target implementation. The success rate refers to $(\#\text{ correctly predicted } \pm\hat{s}_{[i]}/n')$ with respect to the given ν, where n' is 128 for Kyber and 256 for Dilithium.
★ Number of shares: $d=1$ (black), $d=2$ (blue), $d=3$ (red), $d=4$ (teal).

coefficients in our experiments. For Kyber, the (HO)CPAs achieve full success with $\nu=37$, $\nu=900$, $\nu=7000$, $\nu=40000$ traces for $d=1$, $d=2$, $d=3$ and $d=4$, respectively. In the case of Dilithium, full success is observed with $\nu=35$, $\nu=350$, $\nu=1300$, $\nu=4400$ for the corresponding masking orders. The reported success rates are based on (HO)CPA attacks conducted at the PoI identified using the methodology described in the previous paragraph.

7.2.5.5. Post-Processing with lattice attack

Recall from Section 4.4 that recovering $\Upsilon=38$ NTT domain coefficients out of 128 is needed for Kyber768 by using the approach of Kuo & Takayasu (2023). We also adapted the methodology of Kuo & Takayasu (2023) to Dilithium3 and experimentally found out (by performing 100 experiments with uniformly random data) that $\Upsilon=130$ coefficients out of 256 are sufficient when BKZ-20 is used. The runtime of BKZ in the mentioned scenarios and our test equipment is roughly 30 seconds and 19 seconds for Kyber768 and Dilithium3, respectively. Our approach was to employ a block-size for which the overall post-processing using BKZ finishes in practical time based on the results in the literature and in-depth optimization of this step is out of the scope of our work. A critical aspect in applying the lattice attack is determining which NTT domain coefficients have been correctly predicted,

as only these should be included in the process. That is to decide a subset of $\{0, 1, .., n' - 1\}$ of length Υ. If any of the input coefficients is incorrect, the lattice attack fails (the output is larger than η in absolute value) and it must be executed with another subset of coefficients. For this purpose, we make use of the correlation scores of the attacks outputs for each $\hat{s}_{[i]}$. Formally, let $\{\hat{s}'_{[0]}, \hat{s}'_{[1]}, .., \hat{s}'_{[n'-1]}\}$ denotes the predicted set of NTT domain secret coefficients by the (HO)CPA attacks. Let $\Omega = \{\Omega_0, \Omega_1, .., \Omega_{n'-1}\}$ be the corresponding set of confidence scores, where each $\Omega_{[i]} = \mathsf{Score}_i(\hat{s}'_{[i]})$ reflects the score assigned to $\hat{s}'_{[i]}$ during the attack on $\hat{s}_{[i]}$. We then sort Ω in descending order to obtain Ω' and define $\Gamma = \mathsf{argsort}(\Omega)$ as the permutation of coefficient indices that sorts the scores. Starting from the subset of size Υ with the highest-scoring predictions according to Ω, we incrementally consider less likely subsets (as ranked by Ω) until the lattice attack succeeds. Figure 7.7 illustrates an example of this process based on our experimental results. We then sort Ω in descending order to obtain Ω'. Starting from the subset of size Υ with the highest-scoring predictions according to Ω', we incrementally consider less likely subsets (as ranked by Ω) until the lattice attack succeeds. Table 7.4 presents the overall results of the executed (HO)CPA combined with the lattice attack against Kyber and Dilithium. Notice that only **2400** traces are needed in our experiments to break Dilithium with $d = 4$.[5] Notice that, in all cases, the lattice attack recovers the entire s with less than half of the number of traces that the HOCPA requires to achieve full security.

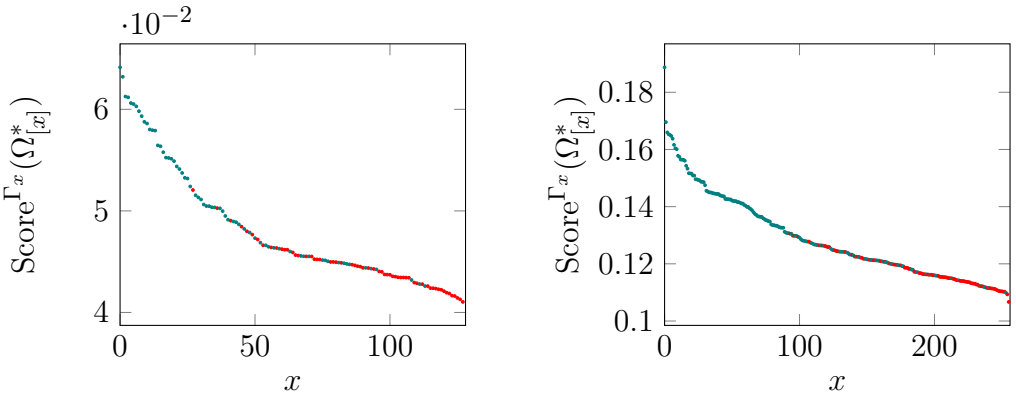
7.2.5.6. Comparison to literature

While no work performs non-profiled SCA attacks for $d > 2$ against the NTT multiplication of Kyber and Dilithium, the attack presented by Dubrova et al. (2023) can be considered an alternative to ours for Kyber, since profiling is done directly on the victim device. Although we acknowledge that the authors do not aim to aggressively minimize the number of traces, they report a total of $\nu = 40K$ traces to break masked Kyber for $d = 3$ and $d = 4$, which is significantly higher than our trace requirements. Moreover, unlike (Dubrova et al., 2023), our approach does not require sending chosen ciphertexts to the victim device.

[5] In the case of Dilithium, for attack orders $d = 2$ and $d = 4$, it was not possible to distinguish the secret from its additive inverse because the prediction functions used are symmetric (see Figure 6.6d). Therefore, we assume that the attacker obtains this sign information from another point in the SCA traces. For Kyber, we did not need to make this assumption.

Table 7.4 Number of traces ν and the amount of time required for the lattice attack successfully retrieve s in our experiments. Results are presented for distinct values of d in the target implementations of both Kyber768 and Dilithium4. Note that ν affects the number of known NTT domain coefficients, denoted by $\#\hat{s}_{[i]}$, obtained through the SCA attack, which serve as inputs to the lattice attack. We report the minimum ν that allows the subsequent lattice attack finishes in practical time, *i.e.* under a day. #LA denotes the number of executions of the lattice attack (with different subset of NTT domain coefficients) until it succeeds. ν_{CPA} denotes the number of traces the (HO)CPA attack achieved 1.0 success rate in our experiments.

Algorithm	d	ν_{CPA}	ν	$\#\hat{s}_{[i]}$	#LA	**Time (m)**
Kyber768	1	37	21	92	4	2
Kyber768	2	900	380	65	700	350
Kyber768	3	7000	2200	59	64	32
Kyber768	4	40000	14500	67	70	35
Dilithium3	1	35	25	243	1	0.31
Dilithium3	2	350	190	201	5	1.58
Dilithium3	3	1300	700	198	1	0.31
Dilithium3	4	4400	2400	198	782	247.6



(a) Kyber, $\nu = 14500$ (b) Dilithium, $\nu = 2000$

Figure 7.7 The sorted set of scores for different coefficient indexes, Ω', for HOCPA with $d = 4$. Correctness of $\hat{s}'_{[i]}$ is visualized, $\hat{s}'_{[x]} = \hat{s}_{[x]}$ in green, otherwise in red. For instance for Dilithium, $\hat{s}'_{[235]}$ led to $\Omega_{[235]} = 0.19$ correlation score during attack on $\hat{s}_{[235]}$. This correlation value is the highest among the set of scores Ω. Therefore its rank is the smallest, $\Gamma_{[0]} = 235$.

8. ATTACKING NON-HW LEAKAGE

This chapter is based on Section 4 and Section 5.2 of the publication (Tosun et al., 2025).

We study the scenario where the device leakage model is not Hamming weight based and unknown to the attacker. To perform first and higher-order SCA attacks in this scenario, we leverage generic SCA distinguishers (see Section 3.5.4). A key challenge here is the injectivity of modular multiplications in NTT based polynomial multiplication, typically addressed by bit-dropping in the literature. However, we experimentally show that bit-dropping is largely inefficient against protected implementations of LBC. To overcome this limitation, we present a novel two-step attack to Kyber, combining generic distinguishers and lattice reduction techniques. Our approach decreases the number of predictions from q^2 to q and does not rely on bit-dropping. Our experimental results demonstrate a speed-up of up to **23490**$\times$ in attack run-time over the baseline along with improved success rate. In certain scenarios, higher-order attacks become feasible only through the proposed approach, as classical methods are shown to be unsuccessful.

8.1. A Novel Attack On Incomplete NTT

In this section, we explain a novel two-step attack strategy targeting incomplete NTT-based polynomial multiplication, such as Kyber's NTT or certain optimized implementations of Dilithium (Abdulrahman et al., 2022). Recall from Chapter 4 that, when attacking the base multiplication in the incomplete NTT domain, the secret coefficients are predicted in pairs of the form $\{\hat{s}_{[i][0]}, \hat{s}_{[i][1]}\}$.

Our attack strategy is as follows. We consider each attacked pair as $\{\hat{s}_{[i][0]}, \hat{s}_{[i][0]}\delta_i\}$. Then, we efficiently retrieve each δ_i using generic SCA distinguishers. Once the full

vector $\Delta = [\delta_i]_{i=0}^{n/2-1}$ is obtained, we perform a lattice attack to recover the even-degree secret coefficients $\hat{s}_{[:][0]}$, effectively reconstructing the full secret. We would like to underline that since our attack employs generic distinguishers, the attacker does not need to predict a device leakage function.

8.1.1. SCA to find Deltas

In this section, we explain how to efficiently find out the relationship between secret coefficients within a pair, *i.e.* δ_i, as defined previously. We re-write the formula for the base multiplication for the higher-order term (see Equation (3.36)) as:

$$\begin{aligned} \hat{r}_{[i][1]} &= \hat{c}_{[i][0]} \cdot \hat{s}_{[i][0]} \cdot \delta_i + \hat{c}_{[i][1]} \cdot \hat{s}_{[i][0]} \pmod q \\ &= \hat{s}_{[i][0]} \cdot (\hat{c}_{[i][0]} \cdot \delta_i + \hat{c}_{[i][1]}) \pmod q \end{aligned} \tag{8.1}$$

As the modular multiplication with a prime modulus is an injective function, $\hat{s}_{[i][0]}$ can't be distinguished through side-channel attacks with generic distinguishers. In particular, for a fixed value of δ_i, the all the elements in set of hypotheses for $\hat{s}_{[i][0]}$, which correspond to $\mathbb{Z}_q^*$, get exactly the same score. The attacker can take the advantage of this feature to retrieve δ_i, by considering the following set of hypotheses $\{(1,1),(1,2),..,,(1,q-1)\}$ for $(\hat{s}_{[i][0]}, \delta_i)$. Note that δ_i can be distinguished for a fixed value of $\hat{s}_{[i][0]}$ since $\hat{c}_{[i][0]} \cdot \delta_i + \hat{c}_{[i][1]} \pmod q$ is non-injective. The prediction with the highest score is expected to be $(1, \delta_i)$.

We should underline that the explained attack must be carried out without using a prediction function (which is meaningful when a generic distinguisher is used). For instance, the attack will not work if the distinguisher is MIA, and it is used with the Hamming weight prediction function.

Notice that such an approach decreases the number of hypotheses from q^2 to q for the studied incomplete NTT based polynomial multiplication, and without the need for traces with specially chosen ciphertexts.

8.1.2. Lattice Attack Using Deltas

In this section, we construct a lattice attack using the knowledge of Δ from the previous section to find out $\hat{s}_{[:][0]}$. We consider the inverse NTT as a matrix multiplication as described in Section 4.4 (following the approach of Kuo & Takayasu (2023)). Recall that the transformation is defined as $s_{[:][0]} = \mathbf{\Phi}\hat{s}_{[:][0]}$ and for the even-degree and odd-degree coefficients, respectively. Also note that $n' = n/2$.

First, we integrate Δ into $\mathbf{\Phi}$ as follows:

$$\mathbf{\Phi}_\Delta = \mathbf{\Phi}(\mathbf{I}_{n'}\Delta) \tag{8.2}$$

where $\mathbf{I}_{n'}$ denotes the identity matrix of size n'. Consequently, we have $s_{[:][1]} = \mathbf{\Phi}_\Delta\hat{s}_{[:][0]}$. Based on this modification, we construct the following basis:

$$\mathbf{B} = \begin{bmatrix} \mathbf{\Phi}^T & \mathbf{\Phi}_\Delta^T \\ q\mathbf{I}_{n'} & 0 \\ 0 & q\mathbf{I}_{n'} \end{bmatrix} \tag{8.3}$$

The lattice $\Lambda(B)$ contains all the linear combinations of the row vectors in $\mathbf{B}$, $\Lambda(B) = \left\{\sum_{i=0}^{3n'-1}\zeta_i \cdot \mathbf{B}_{[i]} \mid \zeta_i \in \mathbb{Z}\right\}$. Then, $[\hat{s}_{[:][0]} \mid \kappa_0 \mid \kappa_1]$ generates the short vector $[s_{[:][0]} \mid s_{[:][1]}]$ in $\Lambda(B)$, for some κ_0, κ_1. Therefore, finding $\hat{s}_{[:][0]}$ is an SVP in $\Lambda(B)$, which can be efficiently solvable for $n' = 128$, using the well-known lattice reduction algorithms LLL or BKZ.

The lattice reduction algorithm returns false positives along with the actual $[s_{[:][0]} \mid s_{[:][1]}]$. In particular, it returns n' possible solutions. We show that all of these solutions are short and comply with our Δ constraint. Consider the ring $\mathcal{R}_{q,n'}$ and let $s_{[:][0]}(x), s_{[:][1]}(x) \in \mathcal{R}_{q,n'}$ denote the polynomials corresponding to the coefficient vectors $s_{[:][0]}$, $s_{[:][1]}$, respectively. Indeed, these solutions are cyclic rotations of $s_{[:][0]}$ and $s_{[:][1]}$ over $\mathcal{R}_{q,n'}$. Rotation in $\mathcal{R}_{q,n'}$ refers to multiplication by the monomial x^α where α denotes the rotation amount.

Multiplication by x^α preserves the shortness property of $s_{[:][0]}$ and $s_{[:][1]}$. Let $t_0 = s_{[:][0]} \cdot x^\alpha$ and $t_1 = s_{[:][1]} \cdot x^\alpha$. Then for $j \in \{0, 1\}$

$$t_{j[i]} = \begin{cases} s_{[i-\alpha][j]}, & \text{if } i-\alpha \geq 0 \\ -s_{[n'-i-\alpha][j]}, & \text{otherwise} \end{cases} \tag{8.4}$$

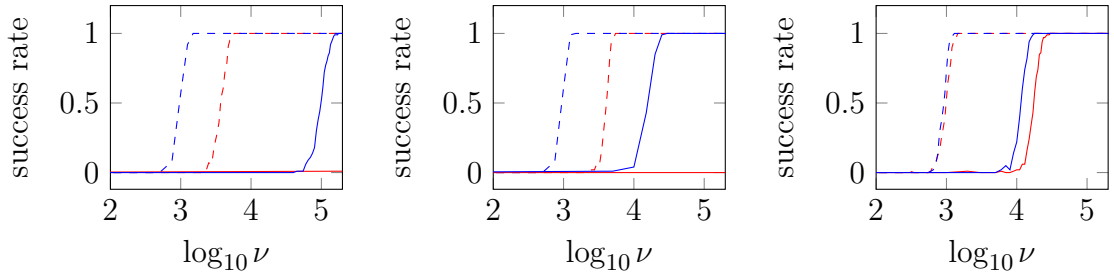
We can consider the Δ constraint in the NTT domain for $\mathcal{R}_{q,n'}$ as $\hat{s}_{[:][1]} = \hat{s}_{[:][0]} \star \Delta$. Taking the inverse NTT of both sides, we get $s_{[:][1]} = s_{[:][0]} \cdot \delta$, where $\delta \in \mathcal{R}_{q,n'}$ denote the polynomial corresponding to the inverse NTT of Δ, with $\delta = \mathbf{\Phi}\Delta$. When both sides are multiplied by x^α, we get $x^\alpha \cdot s_{[:][1]} = x^\alpha \cdot s_{[:][0]} \cdot \delta$. Therefore, $t_1 = t_0 \cdot \delta$, which shows that t_0 and t_1 also satisfy the Δ constraint thus forming valid yet false positive solutions.

Since there are only n' valid solutions as explained above, the attacker can brute-force to distinguish the actual one. Recall that there exists k secret polynomials in the secret key of Kyber $\mathbf{s}$, where k is a parameter based on the target security level, *e.g.* $k = 4$ for Kyber1024. The attacker can efficiently test n' possibilities for each secret polynomial using the public key equation, leading to $(n/2)^k$ possibilities. That is to brute-force 2^{28} possibilities for Kyber1024, and even less for instances of Kyber with lower security levels.

In practice, one can use θ out of n' columns of both $\mathbf{\Phi}^T$ and $\mathbf{\Phi}^T_\Delta$. For instance, when first θ columns are used, $\mathbf{\Phi}^T_{[:][:\theta]}$ and $\mathbf{\Phi}^T_{\Delta[:][:\theta]}$, the lattice reduction returns vectors of size 2θ instead of $2n'$. Note that these vectors are formed by corresponding elements of the form $[s_{[:\theta][0]} \mid s_{[:\theta][1]}]$. While using a smaller θ reduces the complexity of the lattice reduction, aggressively reducing it causes the results to be incorrect, *i.e.* not slices of $[s_{[:][0]} \mid s_{[:][1]}]$. Also note that since the last $n' - \theta$ elements of both $s_{[:][0]}$ and $s_{[:][1]}$ are not retrieved, a complementary second attack is needed. A trivial strategy for this second phase is to target $[s_{[n'-\theta:n'][0]} \mid s_{[n'-\theta:n'][1]}]$ using $\mathbf{\Phi}^T_{[:][n'-\theta:n']}$ and $\mathbf{\Phi}^T_{[:][n'-\theta:n']}$.

8.2. Results

Next, we execute the attack presented in Section 8.1. Since -to the best of our knowledge- there is no publicly available implementation where the device leakage is not a function of Hamming weight, we study this case only through simulations.



(a) Randomly weighted bits

(b) Strongly non-linear leakage

(c) Binary leakage

Figure 8.1 Success rates of first- and second-order SCA attacks on simulated traces generated with different leakage functions, $q = 3329$, SNR $= 1$, with respect to ν. The attack target is to recover δ_i in incomplete NTT domain. For each point in the curves, 100 experiments have been performed with random data. Different distinguishers are used in the SCA attacks.

★ Distinguishers: KW (blue), MIA (red).
★ Number of shares: $d = 1$ (dashed), $d = 2$ (solid).

8.2.1. Simulating Unknown Device Leakage

The simulated traces were generated according to the procedure described in Algorithm 11. Consistent with Section 6.3 and Section 7.2.4, we set $m = 100$ and $\mu = 0$. All experiments were conducted with $q = 3329$ and setting the incomplete NTT arithmetic by $f_1 = 1$ in Algorithm 11. We restrict our analysis to unsigned reduction ($f_2 = 0$), as comparing signed and unsigned modular arithmetic is not relevant in this context. To model unknown device leakage, rather than using only $\mathsf{L} = \mathsf{HW}_\beta$, we also considered the following device leakage functions from Yan et al. (2023) in our experiments:

2.1 Randomly weighted bits: $\mathsf{L}(x) = \sum_{i=0}^{\beta-1} w_i x_{[i]}$ with $w_i \leftarrow [-1, 1]$.
2.2 Strongly non-linear: $\mathsf{L}(x) = \mathsf{B}(x_{[3:0]})$, with B defined to be the Present S-box.
2.3 Binary: $\mathsf{L}(x) = \mathsf{B}(x_{[3:0]}) \mod 2$, with B defined to be the Present S-box.

where $x_{[i]}$ denotes the i-th bit of x and the number of bits in x is denoted by t. $x_{[3:0]}$ denotes the least-significant four bits of x. σ for each set of simulated traces is selected based on the desired signal-to-noise ratio (SNR) for each L. The SNR is defined as $\text{SNR} = \frac{\mathsf{E}[\mathsf{L}(x)^2]}{\mathsf{E}[\sigma^2]}$.

8.2.2. Evaluation of SCA Attack on Deltas

First, we executed the SCA attack on deltas δ_i as explained in Section 8.1.1. As this approach specifically targets incomplete NTT, we evaluate it only for $q = 3329$. We only considered the classical unsigned modular reduction case but the methodology applies to the central reduction case as well. We selected the σ in the simulated traces based on the desired the signal-to-noise (SNR) ratio for each L. As the described methodology requires using a generic distinguisher, we employ both KW and MIA in our experiments. Figure 8.1 demonstrates the results for SCA attack on deltas, revealing the effectiveness of our approach in various leakage characteristics. Although we performed the attacks for $d \in \{1, 2\}$, the method generalizes to higher orders given a sufficient number of traces based on our results. KW outperformed MIA in terms of the number of traces required for a successful attack in our experiments.

8.2.3. Application of Lattice Attack using Deltas

Next, we evaluate the lattice reduction approach proposed in Section 8.1 which aims to recover secret coefficients $\hat{s}_{[i][0]}$ using deltas δ_i obtained from the aforementioned SCA attacks. To validate the approach, we performed 100 experiments with randomly generated s with short coefficients as explained in Section 3.2, considering $\eta = 2$, *e.g.*, Kyber768. For the lattice reduction, we used the BKZ-50 algorithm, which successfully solved the resulting SVP instances for all instances. Through experimentation, we used $\theta = 96$. Our aim here is not to optimize the attack parameters, but rather to demonstrate the feasibility of recovering $\hat{s}_{[i][0]}$ via lattice reduction. In our experimental setup, the lattice reduction process took an average of 884 seconds to complete, with a minimum of 264 seconds, a maximum of 4941 seconds, and a standard deviation of 701 seconds.

Table 8.1 Comparison of attack runtime (in seconds) between the proposed approach and the baseline. Results are demonstrated for $d = 2$ using the distinguishers KW and MIA. Two bins are used to estimate probabilities in the histogram model for MIA.

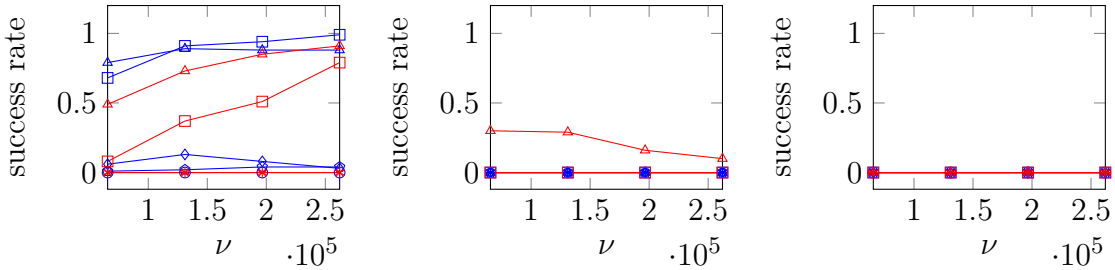
Distinguisher	KW			MIA		
# Dropped bits	4	6	8	4	6	8
Baseline	23490	12900	860	33060	25740	10010
Proposed		10			88	
Speed-up	2349×	1290×	86×	429×	292×	113×

8.2.4. Comparison with Existing Approach

Recall that our approach reduces the number of hypotheses from q^2 to q compared to the baseline approach, which predicts the pair $\{\hat{s}_{[i][0]}, \hat{s}_{[i][1]}\}$ simultaneously (as explained in Section 3.2), and uses bit-dropping. We compare our method to this baseline in terms of attack run-time, demonstrated in Table 8.1 for the distinguishers KW and MIA. For both distinguishers, the speed-up depends on the number of bits dropped. For instance, when half of the bits are dropped, our approach achieves speed-ups of 1290× and 292× speed-up, for KW and MIA, respectively. Determining which bits to drop remains an open question, and typically several configurations are tested. Due to the long run-times required by the baseline method, we omit an in-depth comparison of success rates. However, Figure 8.2 provides evidence that (1) the bit-dropping scheme significantly affects the success rate, and (2) in most bit-dropping configurations and under various device leakage assumptions, the baseline exhibits very low and unstable success rates. In a few specific cases, the baseline achieves promising success rates, but even in those cases, our proposed method performs comparably. In fact, the only scenario where the baseline's success rate approaches 1.0 (though never fully reaching it) is under the device leakage model with randomly weighted bits. Particularly, the accuracy of the baseline with KW and LSB6 is 0.94 for $\nu = 196608$ while it is 1.0 for proposed approach. This analysis is performed as if the NTT is complete to reduce the run-time, which we believe does not impact the conclusion we get from these results.

Additionally, we include experimental results for the unprotected case (i.e., $d = 1$) in Figure 8.3, which serve as evidence that our implementations of KW and MIA

are correct and perform well with bit-dropping when $d = 1$.



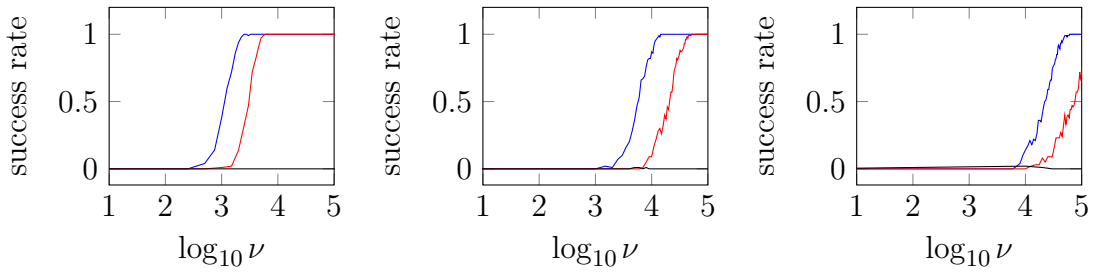
(a) Randomly weighted bits

(b) Strongly non-linear leakage

(c) Binary leakage

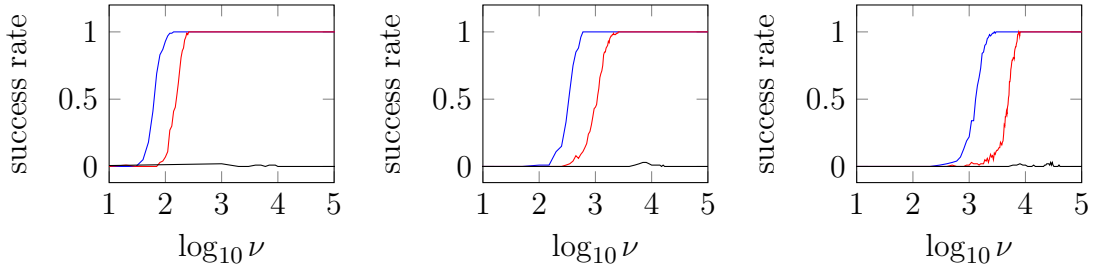
Figure 8.2 Success rates of SCA attacks on simulated traces generated with different device leakage functions, $q = 3329$, SNR$= 1$, $d = 2$, with respect to ν. Different distinguishers and bit-dropping strategies are used in the SCA attacks. For each point in the curves, 100 experiments have been performed with random data.

- ★ Bit-dropping strategies: LSB4 (Least significant 4 bits) (triangle), LSB6 (square), LSB8 (diamond), MSB4 (asterisk), MSB6 (x), MSB8 (+).
- ★ Distinguishers: KW (blue), MIA (red).



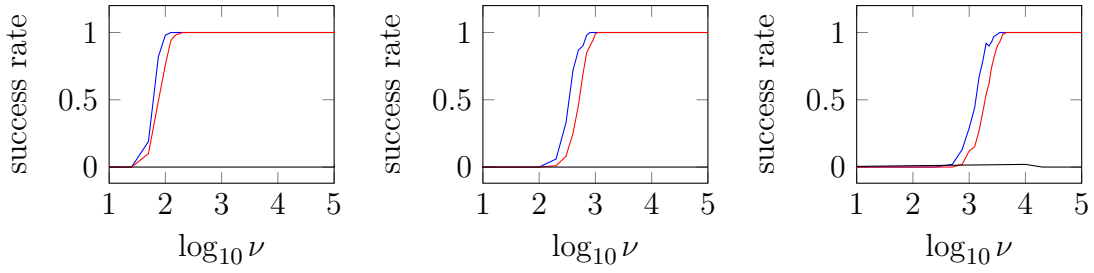
(a.1) SNR $= 1$ (a.2) SNR $= 2^{-3}$ (a.3) SNR $= 2^{-5}$

(a) Randomly weighted bits



(b.1) SNR $= 1$ (b.2) SNR $= 2^{-3}$ (b.3) SNR $= 2^{-5}$

(b) Strongly non-linear leakage



(c.1) SNR $= 1$ (c.2) SNR $= 2^{-3}$ (c.3) SNR $= 2^{-5}$

(c) Binary Leakage

Figure 8.3 Success rates of first-order SCA attacks on simulated traces generated with different device leakage functions, SNRs, $q = 3329$, with respect to ν. For each point in the curves, 100 experiments have been performed with random data.

- ★ Distinguishers: KW (blue), MIA (red).
- ★ Bit-dropping: LSB1 (binary leakage), LSB4 (non-linear leakage), MSB11 (randomly weighted bits).

9. CONCLUSION

This thesis reveals the capabilities of the non-profiled SCA attacks against lattice-based PQC. The study concentrated on the NTT-based polynomial multiplication, as it served as a common and natural target for such attacks. The challenges of adapting existing techniques from the literature to the non-profiled setting in LBC were investigated. To address these challenges, novel solutions were proposed. Both unmasked and masked implementations were studied, including those with higher-order masking. In addition, the scenario in which the leakage function of the device was unknown to the attacker was considered. Overall, the discussions and proposed techniques provided both a technical contribution and a practical guide for conducting non-profiled SCA attacks on LBC implementations.

In Chapter 5, we demonstrated that for incomplete NTT-based implementations of LBC, a speed-up is possible by individually targeting secret coefficients using special public polynomials, specifically, ciphertexts in Kyber and challenge polynomials in Dilithium. In particular, we introduced the zero-value filtering attack, which offers a trade-off between the number of traces and the overall attack run-time. With an appropriate number of traces, this attack can achieve a speed-up of two orders of magnitude over the baseline. Additionally, we proposed an efficient way of verification of predictions on short polynomials, utilizing the inverse NTT transformation. It makes the proposed scheme accurate independent of the number of filtering traces. It was shown that the presented attack is effective in the presence of masking by applying it against a first-order protected implementation of Kyber.

In Chapter 6, we demonstrated that the signed arithmetic that is widely adopted in implementations of LBC such as Abdulrahman et al. (2022); Bronchain & Cassiers (2022); Coron et al. (2024); Heinz et al. (2022); Kannwischer et al. (2019) leads to a vulnerability by making the non-profiled SCA attack significantly easier. State-of-the-art masking LBC, *e.g.* Azouaoui et al. (2023); Beirendonck et al. (2021); Bos et al. (2021); Bronchain & Cassiers (2022); Coron et al. (2024); Fritzmann et al. (2022); Heinz & Dreo Rodosek (2023); Heinz et al. (2022); Migliore et al. (2019); Reparaz et al. (2016), concentrated on developing gadgets to handle non-linear op-

erations and considered the linear part of the algorithms such as the polynomial arithmetic as relatively trivial to mask due to its transparency to arithmetic masking (*i.e.* repeating the operation on each share individually). However, our study revealed that the design decisions such as the reduction technique for the linear parts have a significant impact on the exploitability of the associated SCA leakages and hence on the number of traces required for a successful attack. We also efficiently exploited this source of leakage, by introducing the *absolute value prediction function*. We further have showcased our approach targeting a first-order masked implementation of Kyber. As our attack does not require profiling and is successful with only 250 traces (in our experiments and using our measurement setup), we claim that utilization of the central reduction in masked implementations indeed may ease SCA attacks.

In Chapter 7, we demonstrated that certain higher-order masked implementations of LBC can be efficiently exploited using non-profiled attacks, which pose a significant threat. We enabled efficient HOCPA attacks against second- and higher-order masked implementations of LBC, by revisiting the formulation of the optimal prediction function. When signed modular arithmetic is used in the victim device, we provided explicit formulas involving *sin* and *cos* functions. These prediction functions are particularly effective for attacking second- and higher-order masked implementations. Otherwise, when unsigned modular arithmetic is used, the optimal prediction function can be efficiently computed via our recursive formula. Additionally, the results from Chapter 6 on the impact of using signed arithmetic on SCA leakage were extended to higher-order masking scenarios. We performed HOCPA attacks using the proposed prediction functions against open-source masked implementations of Kyber768 and Dilithium3. We recovered the full secret polynomial with 2200 and 14500 traces against Kyber for second- and third-order masking, and 700 and 2400 against Dilithium, respectively.

In Chapter 8, we demonstrated that non-profiled SCA attacks are feasible even when the device leakage function is unknown to the attacker. In this case, a novel two-step attack was presented that combines generic distinguishers such as MIA and KW with lattice reduction algorithms. Our approach is tailored specifically for incomplete NTT as in Kyber. In particular, we showed that recovering deltas δ_i between pairs of secret coefficients of odd and even degree in the incomplete NTT domain is sufficient for full recovery of secret polynomials. Our approach is more efficient than existing techniques as it reduces the brute-force space from q^2 to q and avoids the bit-dropping trick that leads to information loss and poor success rate. We observed a speed-up in attack run-time ranging from 86× to 2349× in our experiments while the success rate of proposed method is significantly better

in most cases and comparable in some cases. In fact, in some cases, higher-order attacks became feasible only by the proposed approach.

BIBLIOGRAPHY

Abdulrahman, A., Hwang, V., Kannwischer, M. J., & Sprenkels, A. (2022). Faster Kyber and Dilithium on the Cortex-M4. In *International Conference on Applied Cryptography and Network Security*, (pp. 853–871). Springer.

Agrawal, D., Archambeault, B., Rao, J. R., & Rohatgi, P. (2002). The EM side—channel (s). In *International workshop on cryptographic hardware and embedded systems*, (pp. 29–45). Springer.

Aikata, A., Basso, A., Cassiers, G., Mert, A. C., & Roy, S. S. (2023). Kavach: Lightweight masking techniques for polynomial arithmetic in lattice-based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 366–390.

Alkim, E., Barreto, P. S., Bindel, N., Krämer, J., Longa, P., & Ricardini, J. E. (2020). The lattice-based digital signature scheme qtesla. In *International Conference on Applied Cryptography and Network Security*, (pp. 441–460). Springer.

Alkim, E., Bilgin, Y. A., Cenk, M., & Gérard, F. (2020). Cortex-M4 optimizations for {R, M}-LWE schemes. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 336–357.

Azouaoui, M., Bronchain, O., Cassiers, G., Hoffmann, C., Kuzovkova, Y., Renes, J., Schneider, T., Schönauer, M., Standaert, F.-X., & van Vredendaal, C. (2023). Protecting Dilithium against leakage: Revisited sensitivity analysis and improved implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, *2023*(4), 58–79.

Azouaoui, M., Bronchain, O., Cassiers, G., Hoffmann, C., Kuzovkova, Y., Renes, J., Schönauer, M., Schneider, T., Standaert, F.-X., & van Vredendaal, C. (2022). Protecting Dilithium against leakage: Revisited sensitivity analysis and improved implementations. *Cryptology ePrint Archive*.

Backlund, L., Ngo, K., Gärtner, J., & Dubrova, E. (2023). Secret key recovery attack on masked and shuffled implementations of CRYSTALS-Kyber and Saber. In *International Conference on Applied Cryptography and Network Security*, (pp. 159–177). Springer.

Barrett, P. (1986). Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In *Conference on the Theory and Application of Cryptographic Techniques*, (pp. 311–323). Springer.

Beirendonck, M. V., D'anvers, J.-P., Karmakar, A., Balasch, J., & Verbauwhede, I. (2021). A side-channel-resistant implementation of saber. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, *17*(2), 1–26.

Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., & Stehlé, D. (2018). CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on*

Security and Privacy (EuroS&P), (pp. 353–367). IEEE.

Bos, J. W., Gourjon, M., Renes, J., Schneider, T., & Van Vredendaal, C. (2021). Masking Kyber: First-and higher-order implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 173–214.

Botros, L., Kannwischer, M. J., & Schwabe, P. (2019). Memory-efficient high-speed implementation of kyber on cortex-m4. In *International Conference on Cryptology in Africa*, (pp. 209–228). Springer.

Brier, E., Clavier, C., & Olivier, F. (2004). Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems-CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings 6*, (pp. 16–29). Springer.

Bronchain, O., Azouaoui, M., ElGhamrawy, M., Renes, J., & Schneider, T. (2024). Exploiting small-norm polynomial multiplication with physical attacks: Application to crystals-dilithium. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, *2024*(2), 359–383.

Bronchain, O. & Cassiers, G. (2022). Bitslicing arithmetic/boolean masking conversions for fun and profit: with application to lattice-based KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 553–588.

Canright, D. & Batina, L. (2008). A very compact "perfectly masked" S-box for AES. In *International Conference on Applied Cryptography and Network Security*, (pp. 446–459). Springer.

Chari, S., Jutla, C. S., Rao, J. R., & Rohatgi, P. (1999). Towards sound approaches to counteract power-analysis attacks. In *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, (pp. 398–412). Springer.

Chari, S., Rao, J. R., & Rohatgi, P. (2002). Template attacks. In *International workshop on cryptographic hardware and embedded systems*, (pp. 13–28). Springer.

Chen, Z., Karabulut, E., Aysu, A., Ma, Y., & Jing, J. (2021). An efficient non-profiled side-channel attack on the crystals-dilithium post-quantum signature. In *2021 IEEE 39th International Conference on Computer Design (ICCD)*, (pp. 583–590). IEEE.

Cooley, J. W. & Tukey, J. W. (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation*, *19*(90), 297–301.

Coron, J.-S., Gérard, F., Lepoint, T., Trannoy, M., & Zeitoun, R. (2024). Improved high-order masked generation of masking vector and rejection sampling in dilithium. *Cryptology ePrint Archive*.

Coron, J.-S., Gérard, F., Trannoy, M., & Zeitoun, R. (2023). Improved gadgets for the high-order masking of Dilithium. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, *2023*(4).

Dubrova, E., Ngo, K., Gärtner, J., & Wang, R. (2023). Breaking a fifth-order masked implementation of Crystals-Kyber by copy-paste. In *Proceedings of the 10th*

ACM Asia Public-Key Cryptography Workshop, (pp. 10–20).

Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., & Stehlé, D. (2018). Crystals-Dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 238–268.

Dworkin, M. J. et al. (2015). SHA-3 standard: Permutation-based hash and extendable-output functions.

D'Anvers, J.-P., Van Beirendonck, M., & Verbauwhede, I. (2022). Revisiting higher-order masked comparison for lattice-based cryptography: Algorithms and bit-sliced implementations. *IEEE Transactions on Computers*, *72*(2), 321–332.

Fritzmann, T., Van Beirendonck, M., Roy, D. B., Karl, P., Schamberger, T., Verbauwhede, I., & Sigl, G. (2022). Masked accelerators and instruction set extensions for post-quantum cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 414–460.

Fujisaki, E. & Okamoto, T. (1999). Secure integration of asymmetric and symmetric encryption schemes. In *Annual international cryptology conference*, (pp. 537–554). Springer.

Gao, S., Marshall, B., Page, D., & Oswald, E. (2020). Share-slicing: Friend or foe? *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 152–174.

Gentleman, W. M. & Sande, G. (1966). Fast Fourier transforms: for fun and profit. In *Proceedings of the November 7-10, 1966, fall joint computer conference*, (pp. 563–578).

Gierlichs, B., Batina, L., Tuyls, P., & Preneel, B. (2008). Mutual information analysis: A generic side-channel distinguisher. In *International Workshop on Cryptographic Hardware and Embedded Systems*, (pp. 426–442). Springer.

Greconici, D. O., Kannwischer, M. J., & Sprenkels, A. (2021). Compact Dilithium implementations on Cortex-M3 and Cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 1–24.

Groß, H., Schaffenrath, D., & Mangard, S. (2017). Higher-order side-channel protected implementations of KECCAK. In *2017 Euromicro Conference on Digital System Design (DSD)*, (pp. 205–212). IEEE.

Heinz, D. & Dreo Rodosek, G. (2023). Fast first-order masked nttru. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, (pp. 127–148). Springer.

Heinz, D., Kannwischer, M. J., Land, G., Pöppelmann, T., Schwabe, P., & Sprenkels, D. (2022). First-order masked Kyber on ARM Cortex-M4. *Cryptology ePrint Archive*.

Heuser, A., Rioul, O., & Guilley, S. (2014). Good is not good enough: Deriving optimal distinguishers from communication theory. In *Cryptographic Hardware and Embedded Systems–CHES 2014: 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings 16*, (pp. 55–74). Springer.

Huang, J., Zhang, J., Zhao, H., Liu, Z., Cheung, R. C., Koç, Ç. K., & Chen, D. (2022). Improved plantard arithmetic for lattice-based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, *2022*(4), 614–636.

Joye, M., Paillier, P., & Schoenmakers, B. (2005). On second-order differential power analysis. In *Cryptographic Hardware and Embedded Systems–CHES 2005: 7th International Workshop, Edinburgh, UK, August 29–September 1, 2005. Proceedings 7*, (pp. 293–308). Springer.

Kannwischer, M. J., Rijneveld, J., Schwabe, P., & Stoffelen, K. (2019). pqm4: Testing and benchmarking NIST PQC on ARM Cortex-M4.

Karabulut, E., Alkim, E., & Aysu, A. (2021). Single-trace side-channel attacks on ω-small polynomial sampling: With applications to NTRU, NTRU prime, and CRYSTALS-DILITHIUM. In *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, (pp. 35–45). IEEE.

Kim, I.-J., Lee, T.-H., Han, J., Sim, B.-Y., & Han, D.-G. (2020). Novel single-trace ml profiling attacks on nist 3 round candidate dilithium. *Cryptology ePrint Archive*.

Koblitz, N. (1987). Elliptic curve cryptosystems. *Mathematics of computation*, *48*(177), 203–209.

Kocher, P., Jaffe, J., & Jun, B. (1999). Differential power analysis. In *Annual international cryptology conference*, (pp. 388–397). Springer.

Kocher, P. C. (1996). Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Annual international cryptology conference*, (pp. 104–113). Springer.

Kuo, Y.-T. & Takayasu, A. (2023). A lattice attack on crystals-Kyber with correlation power analysis. In *International Conference on Information Security and Cryptology*, (pp. 202–220). Springer.

Langlois, A. & Stehlé, D. (2015). Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, *75*(3), 565–599.

Levi, I., Bellizia, D., & Standaert, F.-X. (2019). Reducing a masked implementation's effective security order with setup manipulations: And an explanation based on externally-amplified couplings. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 293–317.

Librabry, J. I. (2020). Application of attack potential to smartcards (version 3.1).

Lyubashevsky, V., Peikert, C., & Regev, O. (2010). On ideal lattices and learning with errors over rings. In *Annual international conference on the theory and applications of cryptographic techniques*, (pp. 1–23). Springer.

Lyubashevsky, V. & Seiler, G. (2019a). NTTRU: truly fast NTRU using NTT. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, *2019*(3), 180–201.

Lyubashevsky, V. & Seiler, G. (2019b). NTTRU: Truly Fast NTRU Using NTT. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 180–

201.

Mangard, S., Oswald, E., & Popp, T. (2008). *Power analysis attacks: Revealing the secrets of smart cards*, volume 31. Springer Science & Business Media.

Marzougui, S., Ulitzsch, V., Tibouchi, M., & Seifert, J.-P. (2022). Profiling side-channel attacks on Dilithium: A small bit-fiddling leak breaks it all. *Cryptology ePrint Archive*.

Migliore, V., Gérard, B., Tibouchi, M., & Fouque, P.-A. (2019). Masking Dilithium: Efficient implementation and side-channel evaluation. In *Applied Cryptography and Network Security: 17th International Conference, ACNS 2019, Bogota, Colombia, June 5–7, 2019, Proceedings 17*, (pp. 344–362). Springer.

Miller, V. S. (1986). *Use of elliptic curves in cryptography*. Springer.

Montgomery, P. L. (1985). Modular multiplication without trial division. *Mathematics of computation*, *44*(170), 519–521.

Mujdei, C., Wouters, L., Karmakar, A., Beckers, A., Bermudo Mera, J. M., & Verbauwhede, I. (2024). Side-channel analysis of lattice-based post-quantum cryptography: Exploiting polynomial multiplication. *ACM Transactions on Embedded Computing Systems*, *23*(2), 1–23.

Plantard, T. (2021). Efficient word size modular arithmetic. *IEEE Transactions on Emerging Topics in Computing*, *9*(3), 1506–1518.

Primas, R., Pessl, P., & Mangard, S. (2017). Single-trace side-channel attacks on masked lattice-based encryption. In *Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, (pp. 513–533). Springer.

Prouff, E., Rivain, M., & Bevan, R. (2009). Statistical analysis of second order differential power analysis. *IEEE Transactions on computers*, *58*(6), 799–811.

Qiao, Z., Liu, Y., Zhou, Y., Shao, M., & Sun, S. (2023). When NTT meets SIS: Efficient side-channel attacks on Dilithium and Kyber. *Cryptology ePrint Archive*.

Ravi, P., Roy, S. S., Chattopadhyay, A., & Bhasin, S. (2020). Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. *IACR transactions on cryptographic hardware and embedded systems*, 307–335.

Regev, O. (2009). On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, *56*(6), 1–40.

Reparaz, O., Roy, S. S., De Clercq, R., Vercauteren, F., & Verbauwhede, I. (2016). Masking Ring-LWE. *Journal of Cryptographic Engineering*, *6*(2), 139–153.

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, *21*(2), 120–126.

Rodriguez, R. C., Bruguier, F., Valea, E., & Benoit, P. (2023). Correlation electromagnetic analysis on an FPGA implementation of CRYSTALS-Kyber. In *2023 18th Conference on Ph. D Research in Microelectronics and Electronics (PRIME)*, (pp. 217–220). IEEE.

Schnorr, C.-P. (1990). Efficient identification and signatures for smart cards. In *Advances in Cryptology—CRYPTO'89 Proceedings 9*, (pp. 239–252). Springer.

Seiler, G. (2018). Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography. *Cryptology ePrint Archive*.

Shor, P. W. (1994). Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, (pp. 124–134). Ieee.

Steffen, H., Land, G., Kogelheide, L., & Güneysu, T. (2023). Breaking and protecting the Crystal: Side-channel analysis of Dilithium in hardware. In *International Conference on Post-Quantum Cryptography*, (pp. 688–711). Springer.

Tosun, T., Moradi, A., & Savas, E. (2024). Exploiting the central reduction in lattice-based cryptography. *IEEE Access*.

Tosun, T., Oswald, E., & Savaş, E. (2025). Non-profiled higher-order side-channel attacks against lattice-based post-quantum cryptography. *Cryptology ePrint Archive*.

Tosun, T. & Savas, E. (2024). Zero-value filtering for accelerating non-profiled side-channel attack on incomplete NTT based implementations of lattice-based cryptography. *IEEE Transactions on Information Forensics and Security*.

Wang, R., Brisfors, M., & Dubrova, E. (2024). A side-channel attack on a higher-order masked CRYSTALS-Kyber implementation. In *International Conference on Applied Cryptography and Network Security*, (pp. 301–324). Springer.

Xu, Z., Pemberton, O., Roy, S. S., Oswald, D., Yao, W., & Zheng, Z. (2021). Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of kyber. *IEEE Transactions on Computers*, *71*(9), 2163–2176.

Yan, Y., Oswald, E., & Roy, A. (2023). Not optimal but efficient: a distinguisher based on the Kruskal-Wallis test. In *International Conference on Information Security and Cryptology*, (pp. 240–258). Springer.