Master's thesis

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

Erlend Håkegård

Power Analytic Attacks on CRYSTALS-Kyber

Master's thesis in Communication Technology and Digital Security
Supervisor: Colin Alexander Boyd
Co-supervisor: Jonathan Komada Eriksen
June 2023

NTNU
Norwegian University of
Science and Technology

Erlend Håkegård

Power Analytic Attacks on CRYSTALS-Kyber

Master's thesis in Communication Technology and Digital Security
Supervisor: Colin Alexander Boyd
Co-supervisor: Jonathan Komada Eriksen
June 2023

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

NTNU
Norwegian University of
Science and Technology

Title: Power Analytic Attacks on CRYSTALS-Kyber

Student: Håkegård, Erlend

Problem description:

Quantum computers can break widely used public key cryptosystems, leading to new quantum secure algorithms being standardised. To be secure, these new algorithms need to withstand not only quantum attacks but also classical cryptoanalytical attacks and side-channel attacks.

This thesis assesses the security of one of these new algorithms, CRYSTALS-Kyber, against power analytical side-channel attacks. Firstly, the known power analytical attack performed by Karlov et al. will be recreated. Verifying its validity by performing the attack against widely used implementations of CRYSTALS-Kyber. Next, extending the attack to related protocols such as CRYSTALS-Dilithium will be investigated. The thesis will also assess the effectiveness of known countermeasures against power analytical attacks on CRYSTALS-Kyber.

Approved on: 2023-02-23

Main supervisor: Professor Boyd, Colin Alexander, NTNU

Co-supervisor: Eriksen, Jonathan Komada, NTNU

Abstract

This thesis investigates the effect of power analytic side-channel attacks against the post-quantum key encapsulation algorithm CRYSTALS-Kyber. To achieve this, the previously known correlational power analytic attack performed by Karlov et al. [KdG21] was recreated and improved upon. We implemented Kyber512 in the ChipWhisperer toolchain and performed the attack three times, with the power measurements of 1000, 200, and 50 decryptions.

In all the attacks, We successfully recovered all attempted secret key parameters and verified the attack could fully recover the secret key. The attack is also applicable to all versions of CRYSTALS-Kyber, with a linear increase in the runtime of the attack as the security parameter k of CRYSTALS-Kyber increases. We are also confident that the attack can partially recover the secret key in the corresponding signature scheme CRYSTALS-Dilithium.

Our results show a drastic decrease in the likelihood of false positives in the attack as the number of power recordings used increases. However, we observed a drastic increase in runtime due to large computations, requiring future implementations to consider the tradeoff between fault likelihood and runtime.

We have disproven the suggested countermeasure by Karlov et al. of regenerating the secret key every 50 communication cycles by successfully recovering the secret key with the power recordings of 50 decryptions.

Sammendrag

Denne avhandlingen undersøker effekten av effektanalytiske sidekanalangrep på den kvantesikre nøkkelkapslingsalgoritmen CRYSTALS-Kyber. Vi gjenskapte og forbedret et tidligere korrelasjonsangrep utført av Karlov et al. [KdG21]. Vi implementerte Kyber512 i ChipWhisperer-systemet og utførte angrepet tre ganger, med effektmålinger fra 1000, 200 og 50 dekrypteringer.

Vi gjenopprettet alle hemmelige nøkkelparametere i alle gjennomførte angrep og vi har verifisert at angrepet kan gjenopprette den fullstendige hemmelige nøkkelen. Angrepet er også anvendelig på alle versjoner av CRYSTALS-Kyber, med en lineær økning i kjøretiden basert på størrelsen på sikkerhetsparameteren k. Vi er også sikre på at angrepet kan benyttes til å delvis gjenopprette den hemmelige nøkkelen i den tilsvarende digitale signaturalgoritmen CRYSTALS-Dilithium.

Resultatene våre viser en drastisk reduksjon i sannsynligheten for falske positiver i angrep som bruker et økt antall effektmålinger. Imidlertid observerte vi en betydelig økning i kjøretid i angrep med mange effektmålinger, noe som krever at fremtidige implementasjoner må veie risikoen for falske positiver mot kjøretiden.

Vi har avkreftet det foreslåtte mottiltaket av Karlov et al., som omhandler å regenerere den hemmelige nøkkelen innen hver 50. kommunikasjonssyklus. Dette ble gjort ved å gjenopprette den hemmelige nøkkelen ved hjelp av effektmålinger fra 50 dekrypteringer.

Preface

I want to express my deepest gratitude to my supervisors, Colin and Jonathan, for their invaluable guidance and support throughout the completion of this thesis. I greatly appreciate their expertise, insightful feedback and consistent availability. This has significantly enhanced the quality of my work and I am deeply grateful for their mentorship.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

AES Advanced Encryption Standard.

API Application Programming Interfaces.

CMOS Complementary Metal–Oxide–Semiconductor.

CPA Correlation Power Analysis.

CRT Chinese Remainder Theorem.

DPA Differential Power Analysis.

FFT Fast Fourier transform.

FPU Floating Point Unit.

HAL Hardware Abstraction Layer.

KEM Key Encapsulation Mechanisms.

LWE Learning With Errors.

NIST National Institute of Standards and Technology.

NTNU Norwegian University of Science and Technology.

NTT Number Theoretic Transform.

PC Personal Computer.

PKE Public-Key Encryption.

PQM4 Post-Quantum Crypto Library for the ARM Cortex-M4.

RAM Random Access Memory.

SAD Sum of Absolute Difference.

SCA Side-Channel Attack.

SPA Simple Power Analysis.

Chapter 1
Introduction

Cryptography is a crucial part of modern society in securing online communications, protecting our data, and verifying our identity. This is done by relying on the complexity of certain mathematical problems, such as the integer factorisation problem and the discrete logarithm problem. These problems are the foundation of most of the prevailing cryptographic systems.

This security paradigm is threatened by the rapid advancement in quantum computing, with quantum algorithms such as Shor's algorithm [Sho94] being able to solve these problems exponentially faster than today's classical computers. This shift poses several threats to our personal, infrastructural and societal security by potentially exposing all online communication and compromising sensitive data. Quantum computers capable of executing such algorithms are beyond our technological reach. However, their timeline remains uncertain, underscoring the need for developing secure quantum algorithms.

To be prepared for the future, the National Institute of Standards and Technology (NIST) initiated a competition in 2016 to develop and standardise new quantum-secure algorithms. The competition focused on two categories of cryptographic algorithms: Key Encapsulation Mechanisms (KEM) for secure shared-secret distribution between parties and signature schemes to ensure message integrity and authenticity. As a result, four algorithms were selected for standardisation, with CRYSTALS-Kyber emerging as the sole winner in the KEM category and CRYSTALS-Dilithium being one of the three winning signature schemes [NIST]. The potential exists for additional algorithms to be standardised in the future.

While these winning algorithms have undergone rigorous theoretical security testing through the NIST standardisation project, there has been significantly less research concerning their resilience to side-channel attacks – attacks that exploit the processor's physical properties while the cryptographic algorithm is running. Given the increasing presence of hardware devices in the rise of the Internet of Things era,

the need for side-channel security cannot be overstated.

This thesis aims to analyse the security strength of CRYSTALS-Kyber through a power analytic Side-Channel Attack (SCA). The primary focus is to replicate the attack performed by Karlov et al. [KdG21], analyse the attack's performance with varying amounts of available power data, and verify the effectiveness of the proposed countermeasure by Karlov et al., which involves regenerating the secret key before 50 communication cycles. Further, this work will explore whether the attack applies to all versions of CRYSTALS-Kyber and CRYSTALS-Dilithium, a signature scheme built on the same mathematical principles. This investigation leads to the following research questions:

- Can the correlational power analytical attack performed by Karlov et al. be replicated?
- Can the attack be improved?
- How does the availability of varying amounts of power traces impact the performance of the attack?
- Can the countermeasures proposed by Karlov et al. effectively mitigate the attack?
- Is the attack applicable to the corresponding signature scheme, CRYSTALS-Dilithium?

The remainder of the thesis is divided into five chapters. Chapter 2 provides a comprehensive background to the cryptographic methods utilised in this study. Following that, Chapter 3 introduces the power analytic side-channel attacks that form the basis of our research. The setup used for applying the attack to CRYSTALS-Kyber is detailed in Chapter 4. Our research findings are subsequently presented in Chapter 5. The thesis concludes with Chapter 6.

Chapter 2
Background

2.1 Cryptography

Cryptography, the science of protecting information through the use ciphers, serves multiple functions such as ensuring confidentiality, where only the intended recipient can decipher the secret information, guaranteeing data integrity by verifying that there have been no unauthorised alterations, and providing authenticity by confirming the source of information. The use of cryptography extends back thousands of years, with the earliest recorded instance in 1900 BC when Egyptian scribes used non-standard hieroglyphic symbols to obscure messages on clay tablets [WM21]. Today, cryptographic mechanisms are in use all around us. Whether we access a secure website or sign a digital document, cryptography is instrumental in protecting data in transit, verifying website authenticity, and confirming personal identity and content integrity.

2.1.1 Cryptographic schemes

Encryption schemes are used to provide confidentiality to a plaintext message. Three algorithms are required to construct an encryption scheme [KL20]. A key generation algorithm **Gen**; an encryption algorithm **Enc**; and a decryption algorithm **Dec**. The

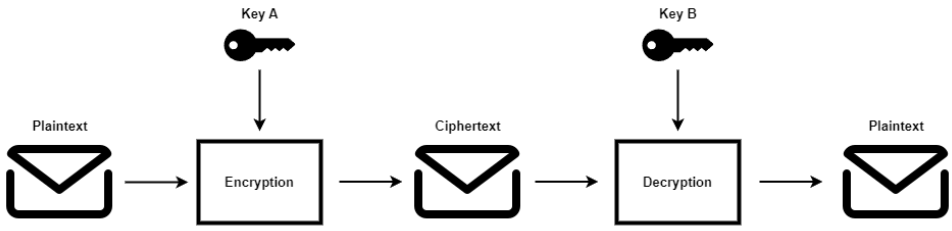


Figure 2.1: Encryption and decryption in a generalized encryption scheme.

general process of sending and receiving encrypted messages is depicted in Figure 2.1. An encryption function first utilises Key A to transmute the plaintext into an unreadable ciphertext, denoted $\mathbf{Enc}_A(Plaintext) = Ciphertext$. Now resistant to prying eyes, this ciphertext can be safely transmitted or stored in untrusted environments. Upon receiving the ciphertext, the intended recipient employs a decryption function, which utilises Key B, to convert the ciphertext back to its original plaintext form, denoted $\mathbf{Dec}_B(Ciphertext) = Plaintext$. It is worth noting that for an encryption process to be deemed secure, it must be infeasible to decipher the ciphertext without knowing the value of Key B. The specific details of these algorithms and the selection of Key A and Key B are determined by the encryption schemes used, with different schemes offering varying levels of security and suited to different use cases. However, the algorithms are commonly divided into symmetric encryption and Public-Key Encryption (PKE).

In symmetric encryption, the same key is used for encryption and decryption, such that $\mathbf{Dec}_K(\mathbf{Enc}_K(Plaintext)) = Plaintext$. In the generalised scheme shown in Figure 2.1, Key A and Key B take the same value. Complete secrecy of the key is imperative for the security of symmetric encryption, as anyone with the key can retrieve the plaintext. This introduces a key distribution problem [KL20]. Both parties involved must have access to the key before the communication starts. Historically, this was addressed by physically meeting to establish the secret keys. However, this is often not feasible, and managing multiple keys can introduce additional risks [KL20].

In a PKE scheme, each participant has a pair of keys: a public key, which can be freely distributed and used by others to encrypt messages, and a private or secret key, which is kept secret and used to decrypt messages. The encryption is done using the recipient's public key, $\mathbf{Enc}_{pk}(Plaintext)) = Ciphertext$, and decryption is performed using the recipient's secret key, $\mathbf{Dec}_{sk}(Ciphertext)) = Plaintext$.

In the generalised scheme in Figure 2.1, Key A is the recipient's public key, and Key B is the recipient's secret key. A significant advantage of PKE schemes is that they eliminate the need for pre-established secrets, as public keys can be safely sent through untrusted environments. However, PKE is typically slower and require bigger keys and ciphertexts than symmetric encryption of the equivalent security strength [KL20].

To mitigate these performance issues, KEM was developed. KEM protocols are used to securely exchange a shared secret key, which can then be used for symmetric encryption. Once both parties have established this shared secret, more efficient symmetric encryption algorithms, such as Advanced Encryption Standard (AES), can be used for subsequent communication. A common method for creating a KEM

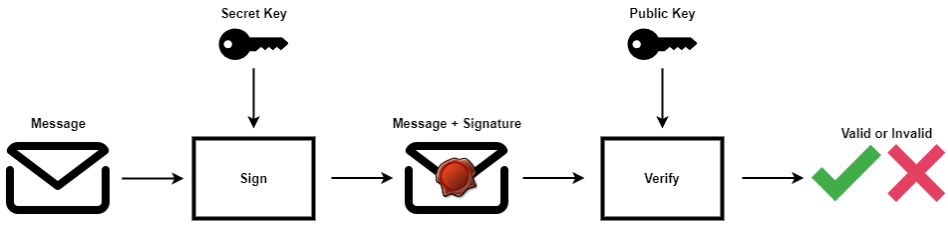


Figure 2.2: The signing and verification process in a generalised signature scheme.

is to modify existing PKE. One method for this conversion is the Fujisaki–Okamoto transform, where PKE schemes secure under chosen plaintext attacks are converted to KEM secure under chosen ciphertext attacks [FO99].

Digital signature schemes are utilised to ensure the integrity and authenticity of a message. Similar to PKE schemes, digital signature schemes also use public and secret keys, but their usage order is reversed. As illustrated in Figure 2.2, to sign a message, the signer employs their private key to compute the signature, which is then distributed along with the message. A verification algorithm subsequently verifies whether the alleged signer has signed a message. The algorithm takes the message and the accompanying signature as input and uses the signer's public key to determine if the signature is valid or invalid [KL20]. If the signature passes the verification process, we can confirm the message's authenticity, as the signer must possess the corresponding secret key to the public key used. Furthermore, we know that the integrity of the message is preserved because any alteration in the message would result in a different signature, causing the verification process to fail.

2.1.2 Quantum computing and cryptography

There is a significant shift in the world of cryptography as quantum computers potentially threaten the security of almost all currently used PKE and signature schemes. The security of most of these schemes is grounded in the computational difficulty of one of two problems: the integer factorisation problem, which involves finding the two prime factors of a large integer, and the discrete logarithm problem, which relies on the difficulty of finding the integer x in the equation $g^x = h$ for a generator g of a cyclic group G. Both of these problems can be efficiently solved in polynomial time using Shor's algorithm on a sufficiently powerful quantum computer [Sho94].

Over the last decade, the field of quantum computing has seen significant advancements, including the public availability of quantum computing through the cloud [IBMQuantum] and the successful, scalable implementation of Shor's algorithm

to factorise the integer 15 [MNM+16]. However, due to technological challenges, it is unknown when quantum computers capable of breaking real-world cryptography will become a reality.

2.2 Lattice problems

CRYSTALS-Kyber is currently the sole winner of the NIST quantum competition for KEM's and will be standardised. This chapter aims to give a fundamental understanding of the algorithm and the concepts it is building upon.

2.2.1 Mathematical background

The evolution of PKE schemes has been marked by increased complexity in the mathematics they employ. From the high school mathematics required for understanding the integer factorisation problem of RSA, leading to the more advanced elliptic curve cryptography and, most lately, to the new lattice-based post-quantum schemes. This section aims to briefly introduce the key mathematical principles of CRYSTALS-Kyber. Atiyah provides a more thorough introduction to mathematical concepts introduced in this section [Ati18].

Ring

A mathematical ring is an algebraic structure consisting of a set of elements, with the two closed operations addition + and multiplication · defined. As in the normal rational numbers, multiplication is required to have an identity element ($a \cdot 1 = a$), be associative ($(a \cdot b) \cdot c = a \cdot (b \cdot c)$) and distributive over addition ($a \cdot (b + c) = a \cdot b + a \cdot c$). However, unlike the rationals, multiplication is not required to be commutative ($a \cdot b \neq b \cdot a$) or have multiplicative inverses defined ($a \cdot a^{-1} \neq 1$). Under addition, a ring is an abelian group, which means there exists an identity element ($a + 0 = a$), there exists an inverse element ($a + (-a) = 0$), it is associative ($(a+b)+c = a+(b+c)$) and is commutative ($a + b = b + a$).

In CRYSTALS-Kyber, a polynomial ring in the form $\mathbb{Z}_q[X]/(f)$ is used. The ring consists of polynomials where the coefficients are integers reduced mod$q$, and the polynomials are reduced mod$f$. The addition of two polynomials is defined coefficient-wise, reduced mod$q$.

$$a + b = \sum_{i=0}^{n-1}((a_i + b_i) \mod q)X^i$$

The multiplication operation is defined as standard polynomial multiplication with the coefficients reduced mod$q$ and the polynomial reduced mod$f$. If f is of the form $X^n + 1$, the ith coefficient of $c = a \cdot b$ is defined as:

$$c_k = \sum_{i+j\equiv k, i+j<n} a_i \cdot b_j - \sum_{i+j\equiv k, i+j\geq n} a_i \cdot b_j$$

Let us consider a numerical example in the ring:

$$R = \mathbb{Z}_{17}[X]/(X^2 - 4)$$

To represent the polynomial $a = 20X^2 + 5X + 17$ in R we need to reduce the coefficents $mod 17$:

$$a = 3X^2 + 5X$$

And reduce the polynomial $\mathrm{mod}(X^2 - 4)$:

$$\begin{aligned} a &= 3X^2 + 5X \\ &= 3X^2 + 5X - 3(X^2 - 4) \\ &= 3X^2 - 3X^2 + 5X - (3 \cdot -4) \\ &= 5X + 12 \end{aligned}$$

The addition is defined coefficient-wise. Let us consider the addition of the polynomial $a = 5X + 8$ and $b = 4X + 9$ in ring R:

$$\begin{aligned} a + b &= (5X + 8) + (4X + 9) \\ &= (5 + 4)X + (8 + 9) \\ &= 9X + 17 \\ &= 9X \end{aligned}$$

The multiplication operation is defined as normal polynomial multiplication before reducing mod f as follows:

$$\begin{aligned} a \cdot b &= (5X + 8)(4X + 9) \\ &= 20X^2 + 45X + 32X + 72 \\ &= 20X^2 + 77X + 72 \\ &= 3X^2 + 9X + 4 \\ &= 3X^2 + 9X + 4 - 3(X^2 - 4) \\ &= 3X^2 - 3X^2 + 9X + 4 - (3 \cdot -4) \\ &= 9X + 16 \end{aligned}$$

Module

A module M is a generalization of a vector space. As in a vector space, the vectors, called elements in the module, form an abelian group. However, unlike vector spaces, the scalars are of a ring R. In finitely generated modules, there exist a set of elements $a_1, a_2, \cdots, a_n \in M$ which reach may reach all elements $x \in M$ by multiplying with scalars $r \in R$.

Lattice

A lattice is a set of infinite discrete points in a vector space. Take the vector space V with the unit vectors v_1, v_2 over the field $\mathbb{R}$. A corresponding lattice Λ would be all integer combinations of v_1 and v_2:

$$\Lambda = \sum_{i=1}^{2} a_i v_i, a \in \mathbb{Z},$$

I.e. Λ is a module over $\mathbb{Z}$, generated by v_1 and v_2.

In CRYSTALS-Kyber, we use modules M over R, where R is the ring from earlier. An element $m \in M$ is a tuple consisting of i polynomials in R. The ring R itself can be seen as a special lattice (with multiplication) reduced mod q, turning M into a so-called module lattice [LS15].

2.2.2 Learning with errors

The Learning With Errors (LWE) problem introduced by Regev [Reg09] has gained considerable attention from cryptographers in recent years due to its potential in post-quantum security. The problem is a reduction of the worst-case lattice problem, which has no known efficient solving algorithms, giving LWE strong security guarantees. In addition, LWE algorithms have proven to be cheap to implement, making it a promising problem for use in embedded devices.

For a formal definition of LWE, some static parameters must be defined. The static parameter $n \geq 1$ denotes the size of the problem, $q \geq 2$ is the size of the modulo and χ is the error probability distribution. $A_{\mathbf{s},\chi}$ is the probobility distrubution for each equation where $\mathbf{a} \in \mathbb{Z}_q^n$ is chosen uniformly at random and $e \in \mathbb{Z}_q$ is choosen acording to the χ distribution. The output of $A_{\mathbf{s},\chi}$ is the sample $(\mathbf{a}, \mathbf{t})$ where $\mathbf{a}$ is one random equation and t is the corresponding equation $\mathbf{a} \cdot \mathbf{s} + e$, where the operations are done modulo q. To solve the LWE problem, an algorithm needs output $\mathbf{s} \in \mathbb{Z}_q^n$ with a high probability given an arbitrary number of samples of $A_{\mathbf{s},\chi}$, the modulo q and the distribution χ for any value of $\mathbf{s}$ [Reg10].

The idea of the LWE problem is to recover a secret message $\mathbf{s}$, represented by a set of random linear equations $\mathbf{A} \mod q$, with each equation having a small additive random error introduced. Take a simple example with the modulo $q = 17$, the secret $\mathbf{s} = (1, 2, 3)$, the error $\mathbf{e} = (1, 0, -1, -1)$ and the random linear equations $\mathbf{A}$:

$$\begin{aligned} &4s_1 + 1s_2 + 6s_3 \\ &5s_1 + 0s_2 + 3s_3 \\ &1s_1 + 1s_2 + 2s_3 \\ &2s_1 + 6s_2 + 5s_3 \end{aligned}$$

For brevity, these equations will herby be represented as matrices:

$$\mathbf{A} = \begin{bmatrix} 4 & 1 & 6 \\ 5 & 0 & 3 \\ 1 & 1 & 2 \\ 2 & 6 & 5 \end{bmatrix}, \mathbf{s} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \mathbf{e} = \begin{bmatrix} 1 \\ 0 \\ -1 \\ -1 \end{bmatrix}$$

$\mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$ is then calculated as follows:

$$\begin{bmatrix} 4 & 1 & 6 \\ 5 & 0 & 3 \\ 1 & 1 & 2 \\ 2 & 6 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 25 \\ 14 \\ 9 \\ 29 \end{bmatrix} \mod 17 = \begin{bmatrix} 8 \\ 14 \\ 9 \\ 12 \end{bmatrix}$$

The LWE problem is then to recover $\mathbf{s}$ knowing only the random equations $\mathbf{A}$ and $\mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$. It is worth to notice without the error recovering the secret message would be trivial after n linearly independent equations, as there is the same number of equations and unknown values.

Many different cryptographic schemes rely on the LWE problem. However, the concepts are the same in hiding the secret message in the error. The following paragraphs show a simple example of such a scheme.

The private key is a uniformly random chosen vector $\mathbf{s} \in \mathbb{Z}_q^n$ while the public key consists of the matrix $\mathbf{A} \in \mathbb{Z}_q^{mxn}$, the the noisy vector $\mathbf{t} \in \mathbb{Z}_q^m = \mathbf{A} \cdot \mathbf{s} + e$ where $\mathbf{e} \in \mathbb{Z}_q^m$ is the error drafted of χ, and the modulo q. This is shown in Figure 2.3 where the public key is denoted in blue and the secret key in green.

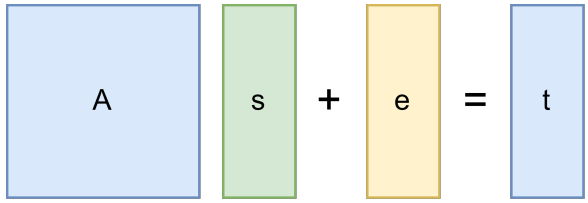


Figure 2.3: The key relationship in the matrix representation of the LWE problem where the public key is shown in blue, the secret key in green and the random error in yellow.

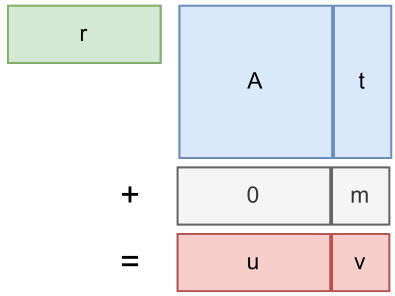


Figure 2.4: The encryption process in a simple LWE scheme. The public key is shown in blue, the secret random vector in green, the encrypted message bit in grey and the ciphertext in red.

The encryption process starts with sampling a random set $r \in \mathbb{Z}_q^m$. The encryption is done bitwise and results in the ciphertexts $\mathbf{u} \in \mathbb{Z}_q^m$ and $v \in \mathbb{Z}_q$. $\mathbf{u}$ is found by multiplying $\mathbf{r}$ and the public key $\mathbf{A}$: $\mathbf{u} = \mathbf{r} \cdot \mathbf{A}$. The value of v depends on the bit to be encrypted. If the message bit is 0, v is the random vector $\mathbf{r}$ multiplied with the public key $\mathbf{t}$. However, if the bit is 1, the half of the modulo is added: $v = \lfloor \frac{q}{2} \rfloor + \mathbf{r} \cdot \mathbf{t}$ [Reg10]. The encryption process is shown in Figure 2.4.

To decrypt a message $v - \mathbf{u} \cdot \mathbf{s} \approx m$ is calculated, as shown in Figure 2.5. If the answer is closer to 0 than $\lfloor \frac{q}{2} \rfloor$ the bit is decrypted as 0, if it is not, the bit is decrypted as 1. It is important to notice that if no error is introduced, the decryption will always be exactly 0 or exactly $\lfloor \frac{q}{2} \rfloor$. A decryption error will occur if the error sum is above $\lfloor \frac{q}{4} \rfloor$. However, this is avoided by carefully setting the standard deviation of χ to appropriate values.

Let us continue by encrypting a 1-bit using the above numerical example.

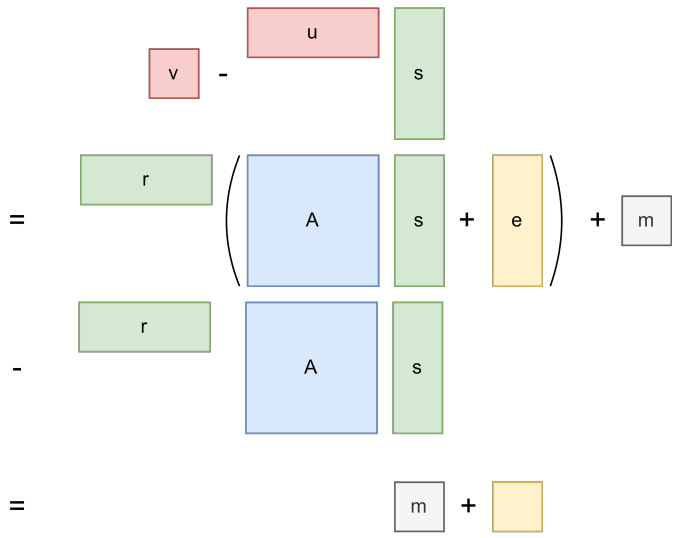


Figure 2.5: The decryption process in a simple LWE scheme. The public key is shown in blue, the secret information in green, the random error in yellow, the encrypted message bit in grey and the ciphertext in red.

We sample the random vector $\mathbf{r} = \begin{bmatrix} 13 & 4 & 7 & 2 \end{bmatrix}$.

$$
\begin{aligned}
\mathbf{u} &= \mathbf{r} \cdot \mathbf{A} \\
&= \begin{bmatrix} 13 & 4 & 7 & 2 \end{bmatrix} \cdot \begin{bmatrix} 4 & 1 & 6 \\ 5 & 0 & 3 \\ 1 & 1 & 2 \\ 2 & 6 & 5 \end{bmatrix} \\
&= \begin{bmatrix} 83 & 32 & 114 \end{bmatrix} \mod 17 \\
&= \begin{bmatrix} 15 & 15 & 16 \end{bmatrix}
\end{aligned}
$$

To calculate v we need to find the value of the message. Since we are encrypting a

1-bit, v is found by calculating:

$$\begin{aligned} v &= \lfloor \frac{q}{2} \rfloor + \mathbf{r} \cdot \mathbf{t} \\ &= \lfloor \frac{17}{2} \rfloor + \begin{bmatrix} 13 & 4 & 7 & 2 \end{bmatrix} \begin{bmatrix} 8 \\ 14 \\ 9 \\ 12 \end{bmatrix} \\ &= 8 + 13 \cdot 8 + 4 \cdot 14 + 7 \cdot 9 + 2 \cdot 12 \mod 17 \\ &= 255 \mod 17 \\ &= 0 \end{aligned}$$

To decrypt $(\mathbf{u}, v)$ the following calculation is made:

$$\begin{aligned} m &\approx v - \mathbf{u} \cdot \mathbf{s} \\ &= 0 - \begin{bmatrix} 15 & 15 & 16 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \\ &= -15 \cdot 1 - 15 \cdot 2 - 16 \cdot 3 \mod 17 \\ &= -93 \mod 17 \\ &= 9 \end{aligned}$$

As 9 is closer to $\lfloor \frac{q}{2} \rfloor = \lfloor \frac{17}{2} \rfloor = 8$ than to 0, the bit was succsessfully decrypted to 1.

The hardness of LWE is based on a quantum reduction from the worst-case lattice problems GAPSVP and SIVP [Reg09], which are believed to be hard as they are widely studied problems with no known efficient quantum algorithms to solve them. A desirable property of the LWE problem is its ability to be reduced to several simpler problems more suited for cryptographic applications. One of the most important is the ability to limit the distribution of the secret $\mathbf{s}$, with minimal changes to the hardness of the problem [Reg10].

One of the main problems of standard LWE cryptographic implementations is the key size. Typically n equations of $A_{\mathbf{s},\chi}$ need to be sent, each of size n, making the key size $O(n^2)$. Ring-LWE introduced by Lyubashevsky et al. [LPR13] resolves this problem by applying algebraic structure while maintaining the strong hardness guarantees of the standard LWE problem. This is done by using the polynomial ring $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ where $q \equiv 1 \bmod 2n$ as the underlying structure of the problem. Similar to standard LWE, there is a secret $s(x) \in R_q$ chosen uniformly at random, an error distribution $e(x) \in R_q$ with small coefficients and m equations,

and an $a(x) \in R_q$ chosen uniformly at random. The noisy representation of $s(x)$ is called $t(x) \approx a(x) \cdot s(x) \in R_q$. The main benefit of Ring-LWE is its efficiency, as the noisy product $t(x)$ produces n pseudorandom values over $\mathbb{Z}_q$ compared to the one scalar obtained in standard LWE [LPR13]. This makes it possible for one sample of $(a(x), t(x)) \in R_q \times R_q$ to pseudorandomly generate n other samples, making the key size $O(n)$.

Let us consider a numerical example in the polynomial ring $\mathbb{Z}_5[X]/(X^2+1)$, where $\mathbf{A}$ consist of the polinomial $a(x) = 4x+3$, the secret polinomial is $s(x) = 2x+4$ and the error polynomial $e(x) = x - 1$. The noisy representation t is calculated as

$$\begin{aligned} t(x) &= a(x) \cdot s(x) + e(x) \\ &= (4x+3)(2x+4) + (x-1) \mod (x^2+1) \\ &= (12x^2 + 16x + 6x + 12) + (x-1) \mod (x^2+1) \\ &= 2x^2 + 3x + 1 \mod (x^2+1) \\ &= 3x - 1 \end{aligned}$$

The ring-LWE problem is then to obtain $s(x) = 2x + 4$ by knowing $a(x) = 4x + 3$ and $t(x) = 3x - 1$.

A generalization of both standard LWE and Ring-LWE was introduced by Brakerski et al. [BGV14] called Module-LWE, which where later proved by Langlois et Al. [LS15] to be as hard as quantumly approximating worst-case lattice problems. In Module-LWE, the underlying algebraic structure is a rank-n module over a polynomial ring denoted as R^n, where the secret $\mathbf{s}$ and error $\mathbf{e}$ are n-tuples of polynomials in R. The sample $(\mathbf{A}, \mathbf{t})$ is generated by choosing $\mathbf{A} \in R^{n \times n}$ uniformly at random and calculating $\mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$ where multiplication and addition are done component-wise in the ring R. The main benefit of module-LWE is increased efficiency compared to unstructured LWE and greater flexibility compared to the ring-LWE because of the extra algebraic structure provided by the module. This allows for more compact cryptographic key sizes and flexible parameter choices, creating the possibility of cryptographic schemes with varying levels of security and efficiency.

For an example, let us consider modules over the polynomial ring $R = \mathbb{Z}_5[X]/(X^2+1)$, where $\mathbf{A} \in R^{2\times 2}$, $\mathbf{s} \in R^2$ and $\mathbf{e} \in R^2$.

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}, \mathbf{s} = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix}, \mathbf{e} = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix},$$

$$
\begin{aligned}
a_{1,1} &= 4X + 2, \\
a_{1,2} &= 2X + 1, \\
a_{2,1} &= X + 3, \\
a_{2,2} &= 4X + 3, \\
s_1 &= 2X + 1, \\
s_2 &= X + 4, \\
e_1 &= X, \\
e_2 &= 4X + 4.
\end{aligned}
$$

The noisy representation $\mathbf{t}$ is then calculated as follows, where all operations are polynomial addition and multiplication in ring R:

$$
\begin{aligned}
\mathbf{t} &= \mathbf{A}\mathbf{s} + \mathbf{e} \\
&= \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \\
&= \begin{bmatrix} a_{1,1}s_1 + a_{1,2}s_2 + e_1 \\ a_{2,1}s_1 + a_{2,2}s_2 + e_2 \end{bmatrix} \\
&= \begin{bmatrix} (4X + 2)(2X + 1) + (2X + 1)(X + 4) + X \\ (X + 3)(2X + 1) + (4X + 3)(X + 4) + 4X + 4 \end{bmatrix} \\
&= \begin{bmatrix} 10X^2 + 18X + 6 \\ 6X^2 + 30X + 19 \end{bmatrix} \\
&= \begin{bmatrix} 0X^2 + 3X + 1 \\ 1X^2 + 0X + 4 \end{bmatrix} \\
&= \begin{bmatrix} 3X + 1 \\ 3 \end{bmatrix}
\end{aligned}
$$

The module-LWE problem is to obtain $\mathbf{s}$ knowing $\mathbf{A}$ and $\mathbf{t}$.

2.2.3 Number theoretic transform

CRYSTALS-Kyber is based on module-LWE and operates on elements in a polynomial ring, which makes polynomial operations a fundamental part of its implementation. Traditional polynomial multiplication methods multiply each term separately and are computationally intensive, requiring $\mathcal{O}(n^2)$ time complexity, where n is the polynomial degree. Given that CRYSTALS-Kyber uses polynomials of degree 256,

and polynomial multiplication is extensively used in key generation, encryption, and decryption [ABL+21], an efficient polynomial multiplication algorithm is essential. In CRYSTALS-Kyber, the Number Theoretic Transform (NTT) have been chosen for polynomial multiplication due to its computational efficiency (requiring only $\mathcal{O}(n \log n)$ operations) and its efficient use of memory and code space [ABL+21].

The main idea of multiplying polynomials through NTT is to represent a large polynomial in a ring as a set of residue polynomials of a lesser degree in a residue class ring. Operations on these smaller polynomials correspond to the same operation on the original larger polynomial. Essentially, the NTT is a special case of the Fast Fourier transform (FFT), which is performed over the field $\mathbb{Z}_q^*$ instead of over the complex numbers [Lyu20]. However, unlike the FFT over the complex numbers, where there always exists an element $\zeta \in \mathbb{C}^*$ of order d (where d is the degree of the product of the two polynomials), such an element is not guaranteed to exist in $\mathbb{Z}_q^*$. The NTT can thus only be applied to multiplication in specific polynomial rings that satisfy certain properties related to the modulus q and the modulus polynomial f [Lyu20]. These properties include the ability to factorize the modulus polynomial f as

$$f(X) = (X - \zeta_1) \cdots (X - \zeta_d)$$

for unique values of $\zeta_i \in \mathbb{Z}_q$. Provided these conditions are met, polynomial multiplication between two polynomials a, b in the ring $\mathbb{Z}_q[X]/f$ can be efficiently performed in three steps [Lyu20]:

1. Reduce the polynomials a, b to the form $\hat{a} = (a_1, \cdots, a_n), \hat{b} = (b_1, \cdots, b_n)$, where each a_i and b_i are residues modulo the factors of f.
2. Perform pointwise multiplication of the residue pairs to obtain $\widehat{ab} = (a_1 b_1, \cdots, a_n b_n)$.
3. Use the Chinese Remainder Theorem (CRT) to reconstruct the product polynomial c from its residues $\hat{c} = \widehat{ab}$.

Let us again consider the multiplication of the polynomials $a = 5X + 8$ and $b = 4X + 9$ in the ring $\mathbb{Z}_{17}[X]/(X^2 - 4)$. The traditional approach would involve $n^2 = 2 \times 2 = 4$ operations:

$$\begin{aligned} 8 \times 9, \\ 8 \times 4X, \\ 5X \times 9, \\ 5X \times 4X. \end{aligned}$$

When NTT is used, we first note that $(x^2 - 4)$ can be factorized as $(x - 2)(x + 2)$. This is a static value which only needs to be calculated once for the ring. We represent

the polynomials a and b in the two rings $\mathbb{Z}_{17}[X]/(X-2)$ and $\mathbb{Z}_{17}[X]/(X+2)$, reducing the polynomials in each ring. As the degree of the reduction polynomial is halved at each step, we require $\log_2 n = \log_2 2 = 1$ step to reduce our degree 1 polynomial to degree 0 (constants):

$$\begin{aligned}
a_1 &\equiv 5X + 8 \bmod (X-2) \equiv 5X + 8 - 5(X-2) \equiv 18 \equiv 1,\\
a_2 &\equiv 5X + 8 \bmod (X+2) \equiv 5X + 8 - 5(X+2) \equiv -2 \equiv 15,\\
b_1 &\equiv 4X + 9 \bmod (X-2) \equiv 4X + 9 - 4(X-2) \equiv 17 \equiv 0,\\
b_2 &\equiv 4X + 9 \bmod (X+2) \equiv 4X + 9 - 4(X+2) \equiv 1.
\end{aligned}$$

Thus, we represent the polynomials a and b as $\hat{a} = (a_1, a_2) = (1, 15)$ and $\hat{b} = (b_1, b_2) = (0, 1)$ in the rings $\mathbb{Z}_{17}[X]/(X-2)$ and $\mathbb{Z}_{17}[X]/(X+2)$.

Next, we perform the pointwise multiplication of the residues of a and b, which can be done in $\mathcal{O}(n)$ operations:

$$\widehat{ab} = (a_1 b_1, a_2 b_2) = (1 \times 0, 15 \times 1) = (0, 15)$$

Finally, we use CRT to combine the residues back into the product polynomial c. This is done by first calculating the ring inverses:

$$(x+2) \bmod (x-2) \equiv 4$$

$$(x-2) \bmod (x+2) \equiv 13$$

Then applying CRT:

$$c = 0 \cdot (x+2) \cdot 13 + 15 \cdot (x-2) \cdot 4$$

$$c = 9x + 16$$

In CRYSTALS-Kyber the plynomial ring $\mathbb{Z}_{3329}[X]/(x^{256}+1)$ is used [ABL+21]. However, when $q = 3329$, the reduction polynomial can only be factorized into 1-degree polynomials. This restriction increases the computational cost for the pointwise multiplication in the NTT domain, as the multiplication of two 1-degree polynomials requires four operations compared to the one operation required to multiply two constants.

2.2.4 CRYSTALS-Kyber

CRYSTALS-Kyber is a quantum chosen ciphertext resistant KEM transformed from an underlying PKE scheme using the Fujisaki–Okamoto transformation [ABL+21]. Three different versions of CRYSTALS-Kyber are defined, each with a different

security strength, Kyber512, Kyber768 and Kyber1024. This section thoroughly explains CRYSTALS-Kyber through how the concepts described earlier are used and how the scheme is defined.

CRYSTALS-Kyber uses the computational difficulty of the Module-LWE problem as the basis for its security. Notably, if an attacker manages to solve the LWE problem, they may trivially obtain the secret key $\mathbf{s}$. The polynomials of $\mathbf{A}$, $\mathbf{s}$ and $\mathbf{e}$ are all of the same ring $\mathbb{Z}_{3329}[X]/(x^{256}+1)$ [ABL+21]. The dimensions of these matrices depend on what security parameter k is used, which is the main mechanism for scaling security [ABL+21]. The matrix dimensions of $\mathbf{A}$ being a $k \times k$ with k being 2, 3 and 4 in Kyber512, Kyber768 and Kyber1024 respectively.

A main difference in the LWE used in CRYSTAL-Kyber compared to traditional approaches is the use of binomial noise in the sampling [ABL+21], compared to the standard Gaussian noise [Reg09]. Gaussian noise sampling has been shown to be inefficient [BCNS14] and vulnerable to timing attacks [BHLY16]. As no known attack depend directly on the noise distribution, The Binomial distribution was chosen due to its efficiency and security [ABL+21].

For the LWE error-correcting algorithms, CRYSTALS-Kyber has defined the compression and decompression functions, which take a polynomial input and operates coefficient wise [ABL+21].

$$\mathbf{Compress}_q = \lceil (2^d/q) \cdot x \rfloor mod^+ 2^d$$

$$\mathbf{Decompress}_q = \lceil (q/2^d) \cdot x \rfloor$$

Compared to traditional LWE cryptography, where the public key is $\mathbf{t}$ and $\mathbf{A}$ and the secret key is $\mathbf{s}$, all keys in CRYSTALS-Kyber are stored in the NTT representation of the polynomials, which increases the efficiency in multiplication. In addition, the public key representation of $\mathbf{A}$ is a 32-byte value ρ used to generate the NTT representation of $\mathbf{A}$, drastically decreasing the key size [ABL+21]. The key size in CRYSTALS-Kyber is dependent on the size of the byte representation of $\hat{\mathbf{t}}$ and $\hat{\mathbf{s}}$. As each vector represents k polynomials and each polynomial is represented in 384 bytes [ABL+21], the public key of the PKE scheme is represented in $384k + 32$ bytes and the secret key is represented in $384k$ bytes. The PKE key generation algorithm can be split into the 6 parts shown in Algorithm 2.1.

The encryption process requires the input of a 32-byte plaintext message and a 32-byte random value in addition to the recipient's public key. The encryption can be split into 10 steps shown in Algorithm 2.2. In step 9, the decompression algorithm is used to create the error tolerance gaps. with 0-bits remaining 0, and 1-bits being converted to $\lceil q/2 \rfloor$. In step 10, the compression function is used to limit the ciphertext size.

Algorithm 2.1 Key generation algorithm for the PKE scheme in CRYSTALS-Kyber.

1: Sample the NTT representation $\hat{R}$ of a polynomial $R \in \mathbb{Z}_{3329}[X]/(x^{256}+1)$ uniformly at random from a generator ρ for all elements in $\hat{\mathbf{A}} \in \hat{R}^{kxk}$.
2: Sample $R \in \mathbb{Z}_{3329}[X]/(x^{256}+1)$ from the Binomial distribution B_{η_1} for all elements in $\mathbf{s} \in R^k$.
3: Sample $R \in \mathbb{Z}_{3329}[X]/(x^{256}+1)$ from the Binomial distribution B_{η_1} for all elements in $\mathbf{e} \in R^k$.
4: Calculate the NTT representations $\hat{\mathbf{s}}$ and $\hat{\mathbf{e}}$ from $\mathbf{s}$ and $\mathbf{e}$.
5: Calculate $\hat{\mathbf{t}} = \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$
6: Create the public key and secret key by encoding the polynomials as a byte string. $pk = Encode_{12}(\hat{\mathbf{t}}|\rho), sk = Encode_{12}(\hat{\mathbf{s}})$.

Algorithm 2.2 Encryption algorithm for the PKE scheme in CRYSTALS-Kyber.

1: Decode the byte representation of the public key, revealing $\hat{\mathbf{t}}$ and ρ.
2: Generate matrix $\hat{\mathbf{A}}^{\mathbf{T}} \in \hat{R}^{kxk}$ using ρ
3: Sample $R \in \mathbb{Z}_{3329}[X]/(x^{256}+1)$ from the Binomial distribution B_{η_1} for all elements in $\mathbf{r} \in R^k$.
4: Sample $R \in \mathbb{Z}_{3329}[X]/(x^{256}+1)$ from the Binomial distribution B_{η_2} for all elements in $\mathbf{e}_1 \in R^k$.
5: Sample $R \in \mathbb{Z}_{3329}[X]/(x^{256}+1)$ from the Binomial distribution B_{η_2} for all elements in $\mathbf{e}_2 \in R$.
6: Calculate the NTT representations $\hat{\mathbf{r}}$ from $\mathbf{r}$.
7: Calculate $\mathbf{u}$ by the inverse NTT of $(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \hat{\mathbf{e}}_1$
8: Decode the byte representation of the plaintext m to its polynomial form
9: Calculate v by $NTT^{-1}(\hat{\mathbf{t}} \circ \hat{\mathbf{r}}) + e_2 + Decompress_q(m, 1)$
10: Create the ciphertext by compressing and encoding $\mathbf{u}$ and v to a byte string representation. $Encode_{d_u}(Compress_q(\mathbf{u}, d_u)|Encode_{d_v}(Compress_q(v, d_v)$

The decryption algorithm takes the secret key and the ciphertext byte arrays as inputs and produces the 32-byte message. The process can be summarised in the 7 steps shown in Algorithm 2.3. In step 6, the compression function is used to interpret if a bit should be decrypted to a 1-bit or 0-bit. For all coefficients in the message polynomial, if $v - \mathbf{s}^T\mathbf{u}, 1$ is closer to $\lceil q/2 \rfloor$ than to 0, the bit is interpreted as a 1-bit, and otherwise, it is interpreted to a 0-bit.

The KEM key generation is similar to one used in the PKE with the only difference being the PKE secret key is encapsulated with the public key, the hash of the public key and 32 random bytes [ABL+21]. This results in the secret key size increasing to 1632 bytes. This process is summarised in the 3 steps of Algorithm 2.4.

The encapsulation algorithm, shown in Algorithm 2.5, generates a shared secret which is encrypted and sent to the communication partner. The algorithm takes

Algorithm 2.3 Decryption algorithm for the PKE scheme in CRYSTALS-Kyber.

1: Obtain $\mathbf{u}$ and v by decoding decompressing the ciphertext
2: Obtain the NTT representation $\hat{\mathbf{s}}$ by decoding the secret key byte array
3: Calculate the NTT representation $\hat{\mathbf{u}}$ from $\mathbf{u}$.
4: Calculate $\widehat{\mathbf{s}^T\mathbf{u}}$ through the pointwise multiplication $\hat{\mathbf{s}}^{\mathbf{T}} \circ \hat{\mathbf{u}}$.
5: Calculate $\mathbf{s}^T\mathbf{u}$ by performing the inverse NTT
6: Calculate $Compress_q(v - \mathbf{s}^T\mathbf{u}, 1)$
7: Obtain the original plaintext message by encoding the polynomial to a byte array.

Algorithm 2.4 Key generation algorithm for the KEM in CRYSTALS-Kyber.

1: Generate z consisting of 32 random bytes.
2: Obtain the keys sk' and pk by running the PKE key generation.
3: Encapsulate the secret key $sk = sk'|pk|H(pk)|z$

the recipient's public key as input and produces the byte-encoded ciphertext c and a byte-encoded secret key K. The algorithm uses three hash functions, H, G and KDF, which are initiated as different functions in different implementations of CRYSTALS-Kyber [ABL+21].

Algorithm 2.5 Encapsulation algorithm for the KEM in CRYSTALS-Kyber.

1: Generate the message m consisting of 32 random bytes
2: Set m to the hash value of $H(m)$ producing 256 bytes
3: Set $(\overline{K}, r) = G(m|H(pk))$
4: Generate ciphertext c by encrypting m through Algorithm 2.2 with the recipient public key and the random values r
5: Generate the shared secret K by the hash $KDF(\overline{K}, H(c)$

The decapsulation algorithm, shown in Algorithm 2.6, takes the ciphertext and secret key as input and verifies and produces the shared secret. In addition, it confirms the correctness of the shared secret by re-encrypting the message to ensure the same ciphertext is made. To be indistinguishable in a chosen ciphertext attack, the algorithm produces a different shared secret if the ciphertexts do not match in line 5.

Algorithm 2.6 Decapsulation algorithm for the KEM in CRYSTALS-Kyber.

1: Obtain sk', pk, $H(pk)$ and z from the secret key sk.
2: Generate m' by decrypting the ciphertext c through the PKE decryption algorithm with the secret key sk'
3: Set $(\overline{K}', r') = G(m'|H(pk))$
4: Generate ciphertext c' by encrypting m' through the PKE encryption algorithm with the recipient public key and the random values r'.
5: Compare c and c'. If they are equal, set the share secret $K = KDF(\overline{K'}, H(c)$, if they are not, set it as $K = KDF(z, H(c))$

Chapter 3
Power analytical side-channel attacks

Two major attack categories exist for attacking cryptographic algorithms, theoretical attacks and Side-Channel Attacks (SCA). While theoretical attacks see the algorithms as mathematic equations and try to recover the secrets through mathematical analysis, SCA are more rooted in the physical world. When a cryptographic algorithm runs on a given processor, the internal state of the processor correlates with the physical, measurable properties of the processor, such as execution time, electromagnetic radiation and power consumption. A SCA is a form of attack which exploits these physical properties to recover the secret parameters of the cryptographic algorithms. In general, SCA are often both more effective and easier to perform than traditional theoretical attacks as no deep mathematical knowledge or a large amount of data is required [ZF05].

In a power analytical SCA, secret information is gained through monitoring the power consumption of the device performing cryptographic operations. Power analytical SCA's are one of the most researched forms of SCA's and have been used to break most symmetric and public key systems, including widely used AES [GT03], RSA [Nov02], DES [KJJ99] and elliptic curve [Cor99] cryptographic schemes.

This chapter introduces power analytical SCA attacks. First, we begin with a general classification of SCA which is then applied to power analytical attacks. This is followed by a demonstration of the different classes of power analytic SCA. Subsequently, we explore the theory behind the advanced power analytic attacks. We then introduce the toolchain used in this thesis for executing power analytical attacks. Lastly, a section on countermeasures against power analytic SCA is provided.

3.1 Classification of side-channel attacks

There are many different types of SCA's, commonly categorised in three independent axes: invasive or non-invasive, active or passive, and simple or differential [ZF05].

Invasive vs non-invasive Invasive attacks require direct access to internal components of the processor, such as monitoring the data bus and compromising the secrets when they are routed internally. Non-invasive attacks only require externally available information. The most common SCA's of this category are timing attacks measuring the computation time of the processor, electromagnetic attacks that exploit the processor's electromagnetic radiation, and power-monitoring attacks that exploit the power usage of the processor.

Active vs passive Attacks classified as active interrupt the correct functioning of the processor and observe the results of leaked information. The most common attacks are fault injections, for example, by voltage glitching and clock glitching. Passive attacks observe the behaviour of the processor without altering its state.

Simple vs differential SCA's are divided into simple and differential class attacks based on the method of analysis. Simple SCA are attacks where the output of the SCA directly depends on the operations performed on the target device. An example of a simple SCA is the first recorded SCA, performed by the British Intelligence Agency in 1964 when they broke the cipher used in the Egyptian embassy. This was done by obtaining the initial state of the Egyptian rotor-cipher machine by using a microphone to listen for the number of clicks in the setup phase of the machine [Wri87]. This is also an example of a non-invasive, passive attack. On the other hand, differential SCA exploit a correlation between the operation performed in a target device and the output of the SCA. These correlations are often small and hidden behind the noise in the measurements. Therefore, several independent recordings and a hypothetical model of the target are required to obtain secret information through strong statistical methods.

Power analytical SCA are classified as passive non-invasive attacks, as the attack only monitors the target, which does not alter its behaviour, and no direct access to the processor is required. However, power analytical attacks require access to the device's power usage, often requiring internal access to the device, making power analytical SCA still require close proximity to the target device. As in SCA classifications, power analytical attacks are divided into Simple Power Analysis (SPA) and Differential Power Analysis (DPA) attacks [KJJ99]. Common for all power analytical SCAs are that traces are analysed to gain secret information. A trace is a set of measurements of the power usage of the target during a cryptographic operation, taken over time at a given sampling rate.

In SPA, a single trace is directly interpolated to gain information on the operation performed in the device and other secret information. This is done by linking peaks

in the trace to operations performed in the target processor. In its simplest form, a SPA attack can be used to detect which cryptographic algorithm is used. This can, for example, be done by linking repeating patterns in the trace to known loops in the algorithms, which could be a first step in part of a bigger attack.

More advanced attacks breaking the cryptographic algorithm are also possible through SPA attacks. This is done by reading the sequence of commands executed, which requires knowledge of the specific implementation of the cryptographic algorithm running on the target and the trace signature of relevant instructions. Suppose the cryptographic implementation uses conditional branching based on the value of secret information. In that case, the branching will be revealed in the trace, also revealing information on the value of the secret. Common operations resulting in conditional branching are memory comparisons which abort on mismatch, multipliers leaking Hamming distance, and exponentiators [KJJ99]. SPA is therefore easily mitigated by avoiding using secret information for conditional branching. The main disadvantage of SPA is the high requirement for signal-to-noise ratio, as the power usage of each instruction must be visible in the trace. This requirement is not true for most complex systems, as other components drain and overshadow power consumption. However, SPA have been successfully performed against simple embedded systems [ÖOP03].

DPA is a more powerful class of attacks where statistical methods are used to exploit a correlation between the value of the data being manipulated and the power usage [Sta10]. Compared to SPA, where instructions are analyzed, the data value has a low effect on a single power trace, often being more insignificant than measurement errors and other noise [KJJ99]. Because of the low signal-to-noise ratio, multiple traces are used in the attacks, with the same key but different known inputs (plaintext or ciphertext). DPA uses a divide-and-conquer strategy, where the key is split into multiple sub-keys. The internal structure of the target algorithm is analyzed to create a leakage model which takes the known input and a sub-key. The leakage is then calculated for all possible sub-keys and known inputs. The statistical methods are then used to compare the predicted leaks for each sub-key and the measured traces, revealing the most likely sub-keys. This process is repeated until all subkeys are revealed.

In the traditional DPA attack, first described by Kocher et al. [KJJ99], the leakage model provides 1 bit which is sent over the data bus. The attack requires traces of multiple encryptions with known inputs. As with all DPA attacks, the attack is performed one sub-key at a time. The traces are divided into two groups based on if the leakage model of the known input and a guessed sub-key is 1 or 0, and the average of all traces in each group is calculated. The biggest peak difference in the value of the two groups is used to evaluate the subkey. This process is repeated for all

possible values of the subkeys, with the subkey value with the biggest difference being the best candidate subkey. This process is repeated for all subkeys revealing the whole secret key. Since the power usage of the data bus is, on average, higher when it is known one bit is 1, there will be a peak difference in the 1-group compared to the 0-group during encryption. In comparison, there is no peak difference in the groups when the wrong subkey is used, as the division of traces is correct only half of the time, cancelling out the peak. The biggest peak across the trace is used to evaluate subkeys, as it is unknown where the encryption is happening in the trace. The attack assumes the non-encryption part of the trace is similar for each group providing no peaks. This assumption does not hold in some cases, as ghost peaks larger than the correct peak may be produced, resulting in erroneous subkeys [BCO04].

An alternative differential class attack is the Correlation Power Analysis (CPA) attack, first described by Brier et al. [BCO04]. The leakage model in a CPA attack reveals the Hamming distance produced by a given input and a subkey. The Hamming distance model assumes the data bus has a constant reference state R before the data is loaded on the bus, and the Hamming distance is the number of bits flipped when the data D is loaded onto the bus from the reference state, denoted $H(D \oplus R)$. It is also assumed the power used in flipping a bit from 0 to 1 is equal to flipping a bit from 1 to 0, and there is a linear relationship between the Hamming distance and the power usage of the processor. The statistical method used in the attack is the Pearson correlation coefficient, used to evaluate the linear relationship between two sets. For the two sets X and Y, the Pearson correlation coefficient is written as:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$$

where *cov* is the covariance of the sets, and σ is the standard deviation. To conduct a CPA attack, the power traces of multiple known input encryptions/decryptions are recorded, and the value of the reference state on the target processor R needs to be known. The leakage model calculates the Hamming distance from all inputs using a given subkey. The Person correlation factor is then calculated between the set of Hamming distance of all inputs for a given subkey guess and the set of all power trace values at a given sample point, as it is unknown when the data is on the bus. This is repeated for all subkey guesses and at every time interval. The best subkey candidate is the subkey with the highest correlation factor.

The main benefits of a CPA attack compared to the traditional DPA attack is the avoidance of ghost peaks and requiring fewer traces for a successful result [BCO04]. However, CPA require a more extensive leakage model as the value of all bits on the databus is required to reveal the Hamming distance, compared to the one bit required in the traditional DPA attack.

3.2 Power analytical SCA in practice

We start by giving a few practical attacks in their simplest form before diving down into the more advanced attack in the later chapters of this thesis. This section demonstrates real-life examples of all attack types mentioned earlier. Starting with a SPA attack against a vulnerable password checker, followed by two attacks against the S-box operation of AES using a traditional DPA approach and a CPA attack. All attacks mentioned in this section are taken from the ChipWhisperer tutorials [NewAEa].

3.2.1 SPA against vulnerable password checker

As described earlier, a SPA attack directly interprets the power usage to gain access to secret information. Algorithm 3.1 is an example of a SPA vulnerable password checker. The algorithm stores the true password in plaintext and compares it with a given password one character at a time. When a mismatched character is found, the password checker aborts. This will lead to a deviation in the instruction sequence based on if the correct or incorrect character is given to the password checker.

Algorithm 3.1 SPA vulnerable password checker implementation

```
for(uint8_t i = 0; i < sizeof(correct_passwd); i++){
    if (correct_passwd[i] != passwd[i]){
        passbad = 1;
        break;
    }
}
```

Figure 3.1 shows the power traces of all possible one-character passwords given to Algorithm 3.1, where each line is the power usage of one password. The grey trace clearly follows a different path than the rest of the traces, which means the character used in the construction of the grey trace must be the first character in the password! The next character can be obtained by collecting traces of all two-letter passwords starting with the correct character from the previous step. This process can be repeated until the whole password is broken.

3.2.2 Traditional DPA attack against AES

The main assumption in a DPA attack is that there is a relationship between the manipulated software data values and the system's power usage. To demonstrate this relationship, we compare the average power usage in two sets of encryptions. The first set consists of 50 power traces where the first byte of the plaintext is set

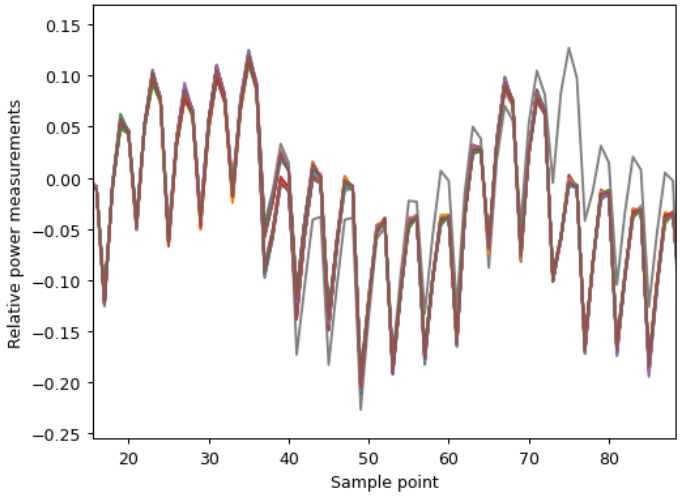


Figure 3.1: Traces of all different one-character password guesses handled by Algorithm 3.1.

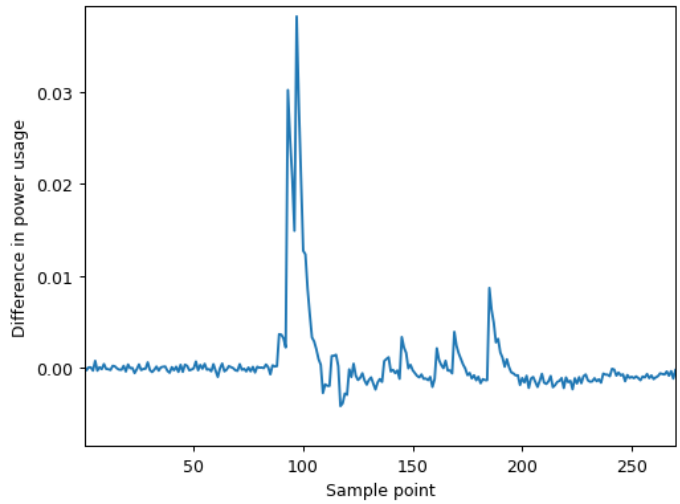


Figure 3.2: Average difference in power usage between encryption of the plaintexts $0xFF$ and $0x00$.

to $0x00$. The second set consists of 50 power traces where the first byte is set to $0xFF$, i.e. all bits are set to 1. The average difference in power usage of the two sets is shown in Figure 3.2, which shows a large peak in power usage difference in the sample point where the all-0 and all-1 bytes are handled.

To exploit this in a traditional DPA attack, we need a leakage model which sorts

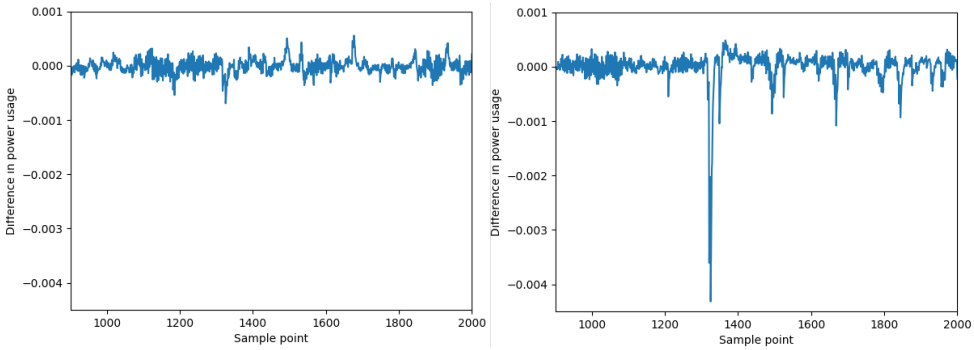


Figure 3.3: Intermediate results of a DPA attack against AES, where **(a)** uses an incorrect key guess, and **(b)** uses the correct key guess.

power traces into two groups based on the value of a specific bit and the value of a guessed key. The difference between these two averages is then compared. When an incorrect key guess is used in the leakage model, the traces are sorted randomly, and there are no peaks in the average power difference. When the correct key guess is used, the groups are divided based on an actual software bit value, which will be visible in the average difference between the two sets of power traces.

Figure 3.3 shows an intermediate result of a DPA attack against the S-box operation of AES. Where the S-box input key byte is guessed, and 1 bit of the output of the S-box operation is the leakage bit. The leakage model calculates the output bit value based on the known ciphertext input byte for all power traces and sorts the traces into two groups based on the value of the bit. The average difference in power usage between the two sets is then calculated, which is shown in Figure 3.3, where 3.3a is the difference when an incorrect key guess is used, and 3.3b is the difference when the correct key guess is used.

3.2.3 CPA attack against AES

The main assumption of CPA attacks is that there exists a linear relation between the Hamming distance of data sent over the databus and the power used by the processor. To demonstrate this relationship, we encrypt 1000 known plaintext with a known secret key and sort the traces in groups based on their Hamming distance. Figure 3.4 shows the average power usage for each Hamming distance group at the sample point where the data is sent over the databus. The figure shows a clear linear relationship, as expected.

Similarly to the DPA attack, a CPA attack can also be used for an attack against

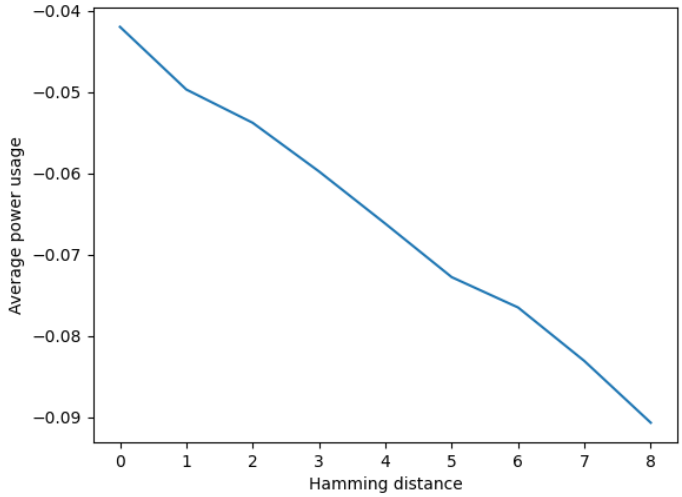


Figure 3.4: Average power usage of groups of Hamming distances.

the S-box of AES. The difference being the leakage model provides the Hamming distance of a theoretical output when given a ciphertext byte and a key guess instead of providing the value of one bit.

In a CPA attack, the Pearson correlation coefficient is used to calculate the linear relationship between two sets. The first set is the set of Hamming distances of all known ciphertext descriptions using the same key guess, and the second set is the power usage of all the decryption processes at a given sample point. The correlation of a key guess is the maximum correlation obtained from the power usage of all sample points.

Figure 3.5 shows the max correlation of all possible key guesses in the attack against AES, with the correct key byte clearly standing out. This process is repeated for all key bytes to obtain the full secret key. Figure 3.6 shows the power usage in the decryption process where each line is a power trace of one decryption, colour coded with the corresponding Hamming distance obtained using the correct subkey. The figure shows a clear linear relationship between the Hamming distance and power usage in the decryption sample point (1325), with darker traces with a high Hamming distance using less power than the lighter traces with a low Hamming distance.

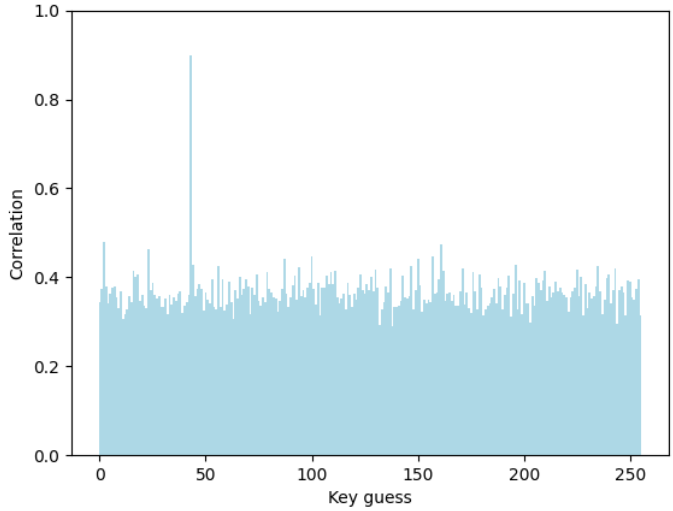


Figure 3.5: Correlation of all key guesses in a CPA attack against AES.

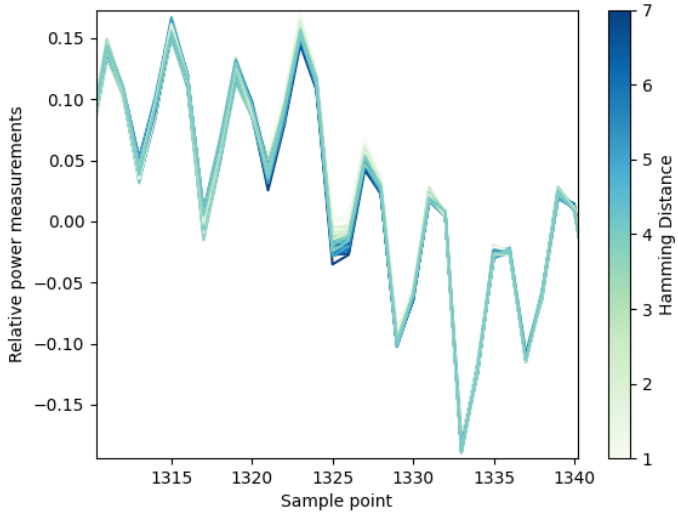


Figure 3.6: All power traces colour coded with the Hamming distance using the leakage model with the correct key at the sample point where the decryption is happening.

3.3 Relationship between databus bit changes and processor power usage

In the previous sections, we have assumed changing bits sent over the databus consumes power. This section provides reasoning for why this is the case.

Currently, most integrated circuits, including processors, are using Complementary Metal–Oxide–Semiconductor (CMOS) gates to execute instructions. A CMOS gate is connected to a power source V_{dd}, ground Gnd, input(s) A and output Q, with the output being dependent on the voltage of the input. The building blocks of CMOS gates are NMOS and PMOS transistors. When the input is high, the NMOS transistor has a low resistance, and the PMOS have a high resistance and vice versa if the input voltage is low.

There are three dissipation sources in CMOS gates, leakage currents in the transistors, short-circuit currents where all transistors are conducting which appear for a short period when A switches, and dynamic power consumption due to load capacitance [Sta10]. In SCA, the dynamic power consumption is the most relevant as it leaks information on the internal state of the integrated circuit [Sta10]. Due to physical properties in the CMOS transistors, the close proximity of the gate and semiconducting material have a natural tendency to store electrical charge and act as a capacitor. When the input state of the CMOS gate changes, the transistors' internal gate capacitance will resist the current change, resulting in dynamic power consumption. Figure 3.7 shows the charge and discharge of the gate capacitance in a CMOS inverter.

The power usage of an integrated circuit depends on the total amount of switching of the gates, which may leak information on the instructions performed as well as more fine-grained information as the data values on the bus. To perform power analytical SCA the power usage needs to be measured. This is done by placing a resistor in series with either the ground or power pin of the processor [KJJ99]. The power is found by using a combination of Ohms and Jouls law by measuring the voltage drop over the resistor $P = IV = \frac{V^2}{R}$.

3.4 ChipWhisperer toolchain

Traditionally conducting power analytical attacks requires a complex setup with a cryptographic device being attacked with an external clock generator and power supply, a power measuring circuit, a sampling oscilloscope and a Personal Computer (PC) to control all measurements [07]. Chipwhispererer is an open-source toolchain which provides all these elements in a standard way, simplifying the process of performing and reproducing power-based SCA and is the toolchain used in our

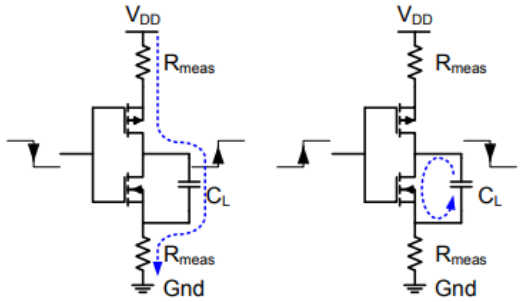


Figure 3.7: The charge and discharge of gate capacitance in a CMOS inverter. The capacitor shown is not part of the CMOS gate, but is a product of the physical characteristics of the integrated circuits. This figure was originally produced by Standaert [Sta10].

attack implementation. This is done through the four components of ChipWhisperer; hardware, firmware, software and tutorials [NewAEa].

3.4.1 Hardware

The ChipWhisperer hardware consists of two categories of circuit boards, scope boards capturing data and target boards being attacked. The communication between the target and capture boards is standardised in a 20-pin connection. ChipWhisperer also provides UFO boards as an intermediary between the scope board and embedded targets to support a magnitude of different target devices.

The scope boards focus on performing power analysis and voltage- and clock-glitching against the target board [NewAEb]. In this thesis, only the power analysis functionality is considered. For power analysis, synchronous sampling is used [OC14], where the sample clock is synchronised with the clock of the target device. This allows only measuring the power on the clock edge, only capturing relevant data and reducing the clock frequency needed for a successful attack. The scope boards have a limited buffer for containing power samples. The Chipwhisperer-Lite used in this thesis has a limit of 24 573 samples. Since it does not support sample streaming [NewAEb], it is more challenging to attack long-duration processes, requiring tradeoffs such as lower sample frequency or splitting the attack in multiple captures. In addition, the scope board acts as the power supply for the target and initiates all communication with the target.

The target boards contain the processor running the code to be attacked with hardware modifications making SCAs more accessible, such as the addition of shunt

resistors and capacitor removal [NewAEb]. Shunt resistors are added to measure the power usage of the target by measuring the voltage drop over the shunt resistor, which gives the current by Ohms law [SBB18] (as described in Section 3.3). Capacitors are a common countermeasure against power analytical attacks, given their property of filtering out noise and stabilising the voltage, hiding the noise generated by cryptographic operations. The target board used in this thesis is the STM32F3 with an ARM Cortex-M4 processor with 256 KB of flash and 40KB of SRAM [NewAEb].

3.4.2 Firmware

The ChipWhisperer firmware provides the program to control the ChipWhisperer hardware components. For the capture boards, the firmware includes logic for handling communication between the host and the target, collecting power traces and initiating glitch attacks against the target board. On the target side, the firmware provides all the necessities for running code on the target board. This includes Hardware Abstraction Layer (HAL) files for each supported target board, which is used for taking a general source code to be built and making it fit the target board hardware architecture. Another part of the firmware is the code running on the target boards. ChipWhisperer provides many different cryptographic schemes already implemented and ready to be built. This code includes the logic handling communication with the software, done through the SimpleSerial protocol, and specifications on where in the code the capture board should start capturing power traces. However, no version of CRYSTALS-Kyber is available and must be integrated from scratch.

The specification of where the capture board should start and stop capturing power traces is done by listening to trigger events. There are several different ways to initiate such events. The most noteworthy is “basic” triggers, where the target notifies the scope directly. Analog triggers send trigger events when power traces match a pre-defined trace within a threshold. And digital triggers where triggering are based on matching patterns on the target pins. In a LAB environment, the “basic” trigger is preferred as we have full control over when the target sends the trigger event. This is done by inserting trigger functions in the firmware running on the target device, which notifies the capture device. When a trigger event is encountered, the scope starts collecting samples of the power usage synchronous to the target’s clock.

The amount of samples collected is set in the software prior to the capture. The samples are stored in the capture buffer, with the size of the buffer being a hard limit on the number of samples the device can store. However, there exist measures to be able to collect longer power traces above the capture buffer size. This can be done by collecting power samples every n clock cycle, which comes with some loss of

information, or by utilizing high-end capture hardware, which allows for streaming of capture data during capture, efficiently removing the length limitation of power traces.

The communication protocol SimpleSerial is the standard protocol for communication between the capture board and the target board. SimpleSerial is a flexible protocol suited to which functionality depends on the firmware running on the target. ChipWhisperer supports two different versions of SimpleSerial, v1.1 and v2.1, with the main difference being the amount of data that can be sent in each packet and cyclic redundancy checks being supported in the latter version. The SimpleSerial communication is always initiated by the capture board with a packet containing an identifying character, the data to be sent to the target and a new line character. Following is an example of a SimpleSerial v1.1 initiation packet:

```
[cmd, data_0, ..., data_n , \n]
```

When the target firmware receives the packet, the data is sent to the function with the same identifier "cmd" as the packet. If the target is to return data to the capture board, a similar packet structure is used. Still, the identifying "cmd" of the returned data has no link to the original identifier. In SimpleSerial v1.1, the data return packet is identical to the original packet shown above. When the target has performed the specified function, an acknowledgement is sent to the capture board containing a status code specified by the firmware of the target. In SimpleSerial v1.1, the acknowledgement has the identifier "z" and the packet is of the following format:

```
['z', status, \n]
```

Given the firmware being open source and having a modular design, it is simple to integrate new functionality in the ChipWhisperer system. New capture and target boards can be integrated by providing new firmware, and new cryptographic schemes can be integrated by providing the source code and program flow logic.

3.4.3 Software

ChipWhisperer also provides a Python library for controlling the hardware and performing SCA's. This is done through four separate Application Programming Interfaces (API). In addition, the ChipWhisperer target devices implement 3rd party interfaces supported by OpenOCS, making it possible to perform real-time debugging on the target device.

The Scope API controls the capture board and provides functionality for setting up, capturing and exporting power traces, setting the trigger the capture device is listening for and setting the clock speed.

The Target API is for controlling the target device under test. It includes functionality to access the bootloaders for the target boards, specifying the firmware to be run on the target, and functionality to communicate with the target board. This communication is often through the SimpleSerial protocol, but other protocols are also supported. When SimpleSerial is used, the target API provides functionality to send SimpleSerial commands to the target and read SimpleSerial commands sent from the target.

The Capture API simplifies the handling of captured traces by attaching relevant information of each trace as the wave, plaintext, ciphertext and the key. Connected traces are also stored through the use of "Projects".

The Analyzer API simplifies the process of performing SCA's. It provides functionality to preprocess all traces in a project through the use of the Sum of Absolute Difference (SAD) algorithm and performing a CPA attack based on a provided leakage model.

3.4.4 Tutorials

The last part of the ChipWhisperer toolchain is the tutorials. Chipwhisperer provides several Jypyter Notebooks containing tutorials for using ChipWhisperer, the API and all its functionality. Several LABs are also provided where the reader is given guidance on performing its own power-analytical and glitching attacks with increased difficulty. The hands-on power analytical examples in Section 3.2 are based on these tutorials. In addition, ChipWhisperer also provides online courses containing the theory of these attacks.

3.5 Countermeasures

For a system to be resistant to power analytical attacks, specific countermeasures need to be implemented. This section gives an overview of such countermeasures, their effectiveness, and their cost.

As mentioned in Section 3.4, a cheap hardware-level countermeasure is integrating capacitors that resist current changes and hide the small power changes exploited by the advanced power analytical attacks. However, these capacitors can easily be removed by an adversary having access to the hardware being attacked, which is often the case in power analytical attacks.

Against SPA attacks, a simple application-level countermeasure is to not alter the program flow based on a secret value. This would prevent large power differences and limit the SPA attack to exploit the instruction sequence. Countermeasures against instruction sequence attacks are to ensure the instructions handling secret information use the same amount of clock cycles regardless of inputs and employ algorithms which do not alter program flow based on inputs. For point multiplication, effective countermeasures have been shown to be algorithms where the addition and double operation patterns are independent of the multiplication [ZF05].

Countermeasures against differential class attacks ass DPA and CPA are more difficult to implement, as they only reduce the pattern differences used in the statistical analysis rather than eliminating them completely [ZF05]. However, a few promising methods have been suggested.

A common countermeasure is randomization, which consists of randomizing the data leaked in the power measurement [ZF05]. As the attacker only obtains randomized data, no statistical methods can be used to obtain the secret information. Examples of such randomization are hardware implementations randomizing the order of instructions [MMS01] or introducing random timing shifts and wait states [ZF05].

Another powerful countermeasure is masking, where the intermediate values processed by an algorithm are masked. At the beginning of the encryption algorithm, the key and the message are masked by a random value, while the rest of the algorithm continues as normal [ZF05]. However, to ensure the correctness of the data sent, mask correction must be applied at the end of the computation, removing the mask resulting in the expected data of the algorithm. In a differential class attack, all patterns are constructed using different masked keys, making the statistical methods unable to recover the true secret key.

In the CPA attack against CRYSTALS-Kyber performed by Karlov et al. [KdG21], the countermeasure of regenerating the private key every 50 communication cycles was suggested. This will limit an attacker to use a maximum of 50 power traces in their analysis, which will allegedly not be sufficient for separating the true keys from the noise. This countermeasure will be evaluated later in this thesis.

Chapter 4
Experimental Setup

This chapter describes the approach undertaken for performing our CPA SCA against CRYSTALS-Kyber. Initially, we describe the architecture of the experimental environment. Subsequently, the attacked CRYSTALS-Kyber implementation is presented. This is followed by a section on our communication system designed for controlling and retrieving data for the attack. Then our procedure for collecting power traces is described. Lastly, an in-depth description of the CPA attack is provided.

4.1 Environment

The hardware devices used in the attack are a ChipWhisperer-Lite capture board, a CW308 UFO board, a CW308T-STM32F3 target device and a PC. See Section 3.4 for a more detailed hardware and software description. The setup is shown in Figure 4.1, where the PC is connected to the capture board through a USB cable, and the capture board is connected to the target board via the UFO board through ChipWhispers standard 20-pin connector for communication and SMA connectors for measuring power.

The PC is running ChipWhisperer software through the provided Jupyter notebooks. The PC is used to build the code on the target device and program the target to run the code. This was done using the build system and bootloaders provided in the ChipWhisperer software. In addition, the PC controls the capture board through standard ChipWhisperer software and the target board using customized SimpleSerial commands, described in Section 4.3.

The ChipWhisperer-Lite capture board is running unaltered firmware provided by ChipWhisperer. It is configured through the OpenADC scope to listen for basic rising edge triggers sent from the target device, to initialize the sampling of power measurements, and to capture 24400 samples just shy of the maximum

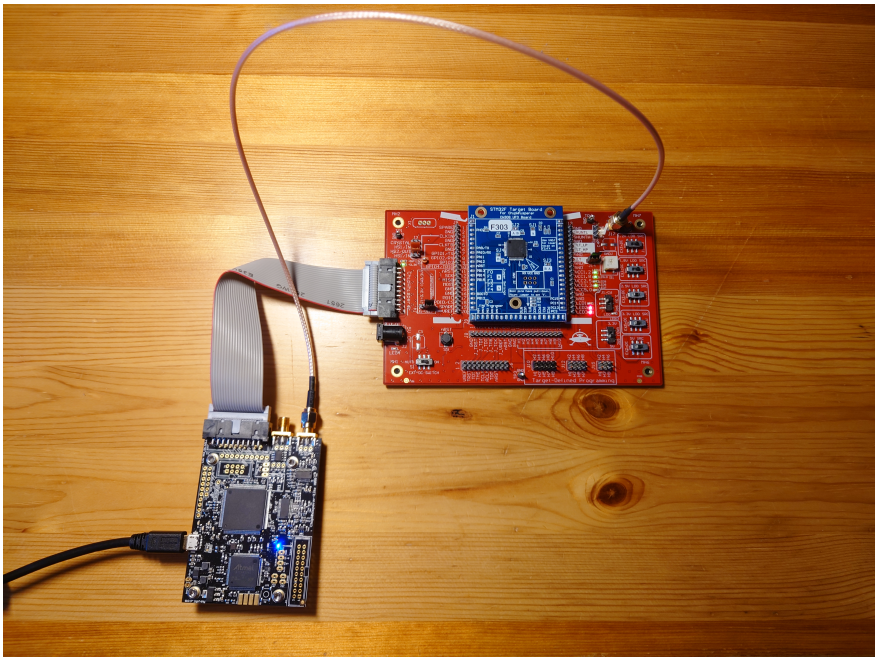


Figure 4.1: Setup of the hardware devices with the PC out of frame, ChipWhisperer-Lite capture board in the bottom left, CW308 UFO board in red and the CW308T-STM32F3 target device on top of the UFO board.

24573 supported by the ChipWhisperer Lite and just enough to capture the whole decryption process in one dimension.

The CW308T-STM32F3 target device runs the Kyber implementation, which will be more thoroughly described in the next section, using its ARM Cortex-M4 processor.

4.2 Kyber implementation

CRYSTALS-Kyber was implemented using the Post-Quantum Crypto Library for the ARM Cortex-M4 (PQM4) [KPR+], which provides implementations of all KEMs and signature schemes involved in the NIST post-quantum competition for the ARM Cortex-M4 family of microcontrollers. However, an earlier version was used due to problems integrating the ARM Cortex-M4 Floating Point Unit (FPU), which is widely used in the latest PQM4 versions, with the ChipWhisperer build system. The CRYSTALS-Kyber version of commit 8970d37 of PQM4, released Sep 26, 2021, was used as it is the latest version not using the ARM Cortex-M4 FPU and still providing a valid implementation of the latest Round 3 submission of CRYSTALS-Kyber.

Given the limited amount of Random Access Memory (RAM) in the CW308T-STM32F3 target device, it is not possible to run the full implementation of CRYSTALS-Kyber as a KEM (described in Section 2.1). Therefore, CRYSTALS-Kyber is implemented as a PKE scheme, directly implementing the algorithms for PKE key generation, encryption and decryption, bypassing all KEM algorithms (see Section 2.2.4). The PKE key generation and encryption algorithms were implemented unmodified from PQM4. As we are attacking the base multiplication of the ciphertext and the secret key (line 4 in Algorithm 2.3), the PKE decryption algorithm was modified to send trigger signals, initiating the power recording on the capture board just before the polynomial multiplication.

Algorithm 4.1 shows the decryption algorithm where the first "trigger_high();" initiates power trace collection for the first dimension, and the second initiates power trace collection for the remaining dimensions.

Algorithm 4.1 Modified PKE decryption algorithm

```
indcpa_dec(unsigned char *m, unsigned char *c, unsigned char *sk) {
    poly mp, bp;
    poly *v = &bp;

    poly_unpackdecompress(&mp, c, 0);
    poly_ntt(&mp);
    trigger_high();
    poly_frombytes_mul(&mp, sk);
    trigger_low();
    for(int i = 1; i < KYBER_K; i++) {
        poly_unpackdecompress(&bp, c, i);
        poly_ntt(&bp);
        trigger_high();
        poly_frombytes_mul(&bp, sk + i*KYBER_POLYBYTES);
        trigger_low();
        poly_add(&mp, &mp, &bp);
    }

    poly_invntt(&mp);
    poly_decompress(v, c+KYBER_POLYVECCOMPRESSEDBYTES);
    poly_sub(&mp, v, &mp);
    poly_reduce(&mp);

    poly_tomsg(m, &mp);
}
```

4.3 Communication system

To conduct the CPA attack against CRYSTALS-Kyber, we need to know when the decryption takes place and what ciphertext is being decrypted. However, in embedded systems, there is no simple way to view and modify the state of the software in runtime. Therefore, a communication system was created between the target device and the ChipWhisperer software. The system is implemented in the firmware running in the target device and as Python scripts in the ChipWhisperer software. The SimpleSerial protocol version 1.1 (described in Section 3.4) was used to relay data between the PC and the target device.

A communication cycle can be summarized in the following steps:

1. The PC initiate the communication through the target API of the ChipWhispere software by sending a SimpleSerial packet with an identifying character. See Section 3.4 for more details.
2. The packet arrives at the target, where the identifying character is interpreted.
3. If a match is found, the corresponding firmware code is initiated.
4. If the target is to return data, it responds with a SimpleSerial packet with an identifying character followed by the message.
5. When the function has been completed, the target returns an acknowledgement packet.
6. The response is obtained in the PC by issuing a read command to the target API.

Function	**Character**	**Respone**
Initiate PKE key generation	k	None
Initiate PKE encryption	e	None
Initiate PKE decryption	d	None
Get public key	p	32 byte of public key
Get secret key	s	32 byte of secret key
Get ciphertext	c	32 byte of ciphertext
Generate and return plaintext	i	Newly generated plaintext
Get decrypted plaintext	o	Output of previous PKE decryption
Reset	r	None

Table 4.1: Overview of all implemented SimpleSerial commands.

Table 4.1 summarises the implemented commands. Given the limited maximum packet size of 64 bytes, retrieving longer variables as keys and ciphertext in one packet is impossible. This is solved by requiring the PC to send multiple requests for the same variable, with each subsequent request being responded to with the next 32 bytes of the variable. A simple counter variable keeps track of what bytes are sent, which may also be reset by sending a SimpleSerial packet with the reset character.

Algorithm 4.2 Base multiplication of 4 bytes, part of the doublebasemul_asm() function.

```
//basemul 2
smultt tmp, poly0, poly1
montgomery q, qinv, tmp, tmp2
smultb tmp2, tmp2, zeta
smlabb tmp2, poly0, poly1, tmp2
montgomery q, qinv, tmp2, tmp

smuadx tmp2, poly0, poly1
montgomery q, qinv, tmp2, tmp3
pkhtb tmp, tmp3, tmp, asr#16
str tmp, [rptr], #4

neg zeta, zeta

//basemul 2
smultt tmp, poly2, poly3
montgomery q, qinv, tmp, tmp2
smultb tmp2, tmp2, zeta
smlabb tmp2, poly2, poly3, tmp2
montgomery q, qinv, tmp2, tmp

smuadx tmp2, poly2, poly3
montgomery q, qinv, tmp2, tmp3
pkhtb tmp, tmp3, tmp, asr#16
str tmp, [rptr], #4
```

4.4 Power trace collection

In the CPA attack, we are interested in the power traces containing measurements of the power usage of the base multiplication between the secret key and ciphertext in the NTT domain. This operation is performed in Algorithm 4.2 in our CRYSTALS-Kyber implementations and which is line 4 of Algorithm 2.3 in the specifications.

We want to determine the relationship between the amount of power traces collected and the attack's efficiency. Therefore, it was decided to record a total of 1000 power traces of the decryption process and use subsets of these traces in different attacks.

Ideally, one power trace should record power measurements of the whole decryption process. However, this is not possible given the low maximum sample size of the ChipWhisperer-Lite of 24573 measurement points. A maximum-size ChipWhisperer-Lite power trace manages to record the decryption in one dimension. Collecting multiple power traces in one setting is infeasible due to the risk of overwriting traces before retrieval. Therefore the decryption of each ciphertext needs to be performed k times to collect the k power traces needed. In our attack against Kyber512, where $k = 2$, we need to decrypt each ciphertext two times. To ensure there are no deviations in each recording, all randomness is obtained using a pseudorandom function with the same initial seed.

Prior to recording the power traces, a static public and secret key were obtained by running the PKE key generation function. This ensures all decryptions are done using the same secret keys. The keys were retrieved through the SimpleSerial communication system to set their value statically in the code.

To obtain the power trace of one decryption, algorithm 4.3 was used. Firstly a new plaintext is generated, stored and encrypted using the communication framework. Next, the scope is armed, making the capture board listen for trigger signals before the decryption is initialized. When the program has received an acknowledgement that decryption is finished, it retrieves the decrypted plaintext and verifies it is the same as the initial plaintext. If the decryption is successful, the trace is collected and stored together with the corresponding plaintext and ciphertext. This process is repeated for all 1000 decryptions and all k dimensions.

4.5 CPA attack implementation

Our CPA attack is a slightly modified version of the attack performed by Karlov et al. [KdG21]. We run three attack variations using 1000, 200 and 50 power traces of known ciphertext decryptions, each trace containing the 24400 sample points. The attack will be explained using 1000 power traces, but the approach is the same in the other attack variations. For the source code implementation of the attack, see the associated repository[1].

As described in Section 3, a CPA attack require us to know the Hamming distance of a word sent over the databus, which depends on the secret value we are attempting

[1] https://github.com/erlehaak/chipwhisperer

Algorithm 4.3 Algorithm for obtaining power trace of one decryption.

```
def get_trace():
  #Generates and saves plaintext to be sent
  plaintext_in = get_pt_i()

  #Encrypt
  target.simpleserial_write('e', bytearray())
  target.simpleserial_wait_ack()

  #Decrypt
  scope.arm()
  target.simpleserial_write('d', bytearray())
  scope.capture()
  target.simpleserial_wait_ack();

  #Get decrypted plaintext
  plaintext_out= get_pt_o()

  #Check if valid trace
  assert plaintext_in == plaintext_out

  ciphertext = get_ciphertext()
  trace = scope.get_last_trace()

  return [plaintext_in, ciphertext, trace]
```

to recover. In our attack, we attempt to recover the secret key by exploiting the base multiplication between the secret key and the ciphertext in the NTT domain, which is shown in line 4 of the decryption algorithm 2.3. A more thorough explanation of the base multiplication is found in section 2.2.3. In each of these base multiplications, the two 1-degree polynomials $s_1 + s_2X$ and $c_1 + c_2X$ are multiplied. Where the coefficients of $s_1 + s_2X$ are dependent on the secret key, and the coefficients of $c_1 + c_2X$ are dependent on the ciphertext.

In the CRYSTALS-Kyber implementation, we attack Algorithm 4.2 where the registers poly0 and poly2 contain one ciphertext polynomial each as $c_1|c_2$, and the registers poly1 and poly3 contain one secret key polynomial each as $s_1|s_2$. Algorithm 4.2 calculates two polynomial multiplications, $poly0 \cdot poly1$ in the top half, and $poly2 \cdot poly3$ in the bottom half. As we perform a known ciphertext attack, all values except for the secret key coefficients s_1 and s_2 are known. The ciphertext polynomial values c_1 and c_2 can be obtained by decoding, decompressing and performing NTT on the ciphertext according to the CRYSTALS-Kyber specification (see section 2.2.4).

The ζ values are static and are available in the implementation-specific source code.

The following paragraphs describe the attack against one polynomial multiplication. For a full attack recovering the whole secret key, this process needs to be repeated 128 times for each dimension k in CRYSTALS-Kyber, which is $2 \cdots 128 = 248$ times for recovering a full Kyber512 secret key.

We divide the attack on each polynomial multiplication into two parts. In the first part, we are interested in the result of the first "smultt" operation, which multiplies the top coefficient of the ciphertext polynomial c_2 with the top coefficient of the secret key polynomial s_2. In the second part, we are interested in the value in the tmp register when it is stored in memory with the "str" instruction. The content of this register is the product polynomial after the base multiplication $(p_1 + p_2X)$, with the top half containing the p_2 coefficient and the bottom part containing the p_1 coefficient. Calculating the Hamming distance of these values depends on the processor. For the CW308T-STM32F3, the data is represented in two's complement, and the data bus is set to all 0 bits before a word is sent. Therefore the Hamming distance is obtained by counting the number of 1-bits in the two's complement representation of the values.

Part 1 of the attack starts with calculating the resulting Hamming distance of the "smultt" operation, which multiplies $c_2 \times s_2$ using all possible values of s_2. Even though 4 bytes are used to store s_2, the CRYSTALS-Kyber documentation specifies the integer values of coefficients are reduced $\bmod q = 3329$, resulting in the calculation of the Hamming distance of 3329 s_2 coefficient guesses for each of the 1000 ciphertexts.

The next step is to find the Person correlation coefficient between the Hamming distance and power usage. We define a set of Hamming distances as the Hamming distance of the result of the "smultt" operation using one secret key coefficient s_2 and all 1000 values of the ciphertext coefficient c_2. As we do not know when the result of the "smultt" operation is sent over the data bus, we calculate the correlation in all sample points. We define a set of power usage as the power usage of all 1000 ciphertext decryption processes at a given sample point. We then calculate the Person correlation coefficient between all combinations between the 3329 sets of Hamming distances and the 24400 sets of power measurements for a total of 81560500 calculations.

To prepare for part 2 of the attack, we store the maximum achieved correlation of all 3329 secret key coefficient s_2 and order them from highest to lowest correlation.

In part 2 of the attack, the process is repeated with the Hamming distance of the "str" result being calculated instead. We start with the secret key s_2 coefficient with

the highest maximum correlation from part 1 of the attack. We then calculate the Hamming distance of the resulting "str" operation using s_2 and all 3329 values of s_1. As in the previous step, the Person correlation coefficient is then calculated between all combinations of the 3329 sets of Hamming distances and the 24400 sets of power measurements. If a correlation higher than a given threshold is encountered, 0.9 in our case, the used s_1 and s_2 coefficients are set as the true secret key coefficient, and the attack moves to the next polynomial multiplication. If no such correlation is found, the attack moves to the s_2 coefficient with the next highest correlation from part 1, and the process is repeated.

The attack will be performed against Kyber512 with three different numbers of known ciphertext power traces, 1000, 200 and 50. Part 1 will be performed against all polynomial multiplications. Part 2 of the attack with the correct value of the s_2 coefficient will also be performed against all polynomial multiplications. However, the full attack will only be performed against select polynomial multiplications:

- The polynomial multiplication where the correct s_2 coefficient has the highest correlation value in part 1.
- The polynomial multiplication where the correct s_2 coefficient has the lowest correlation value in part 1.
- The polynomial multiplication where the correct combination of s_1 and s_2 coefficients has the highest correlation value in part 2.
- The polynomial multiplication where the correct combination of s_1 and s_2 coefficients has the lowest correlation value in part 2.

4.5.1 Improvements

This attack is a recreation of Karlov et al.'s attack [KdG21] with some modifications described in this section.

In the base multiplication algorithm shown in Algorithm 4.2, each polynomial coefficient is stored using 2 bytes. In the original attack, all $2^{16} = 65536$ possible values were evaluated. This is unnecessary as the modulus $q = 3329$ defined in CRYSTALS-Kyber ensures all coefficients are lower than 3329. Therefore only the first 3329 coefficients are evaluated in our implementation, drastically improving the attack's efficiency.

Another difference is the selection of which s_2 coefficients should be evaluated in part 2 of the attack. In the original attack, all s_2 coefficients which achieved a correlation above a given threshold, set to 0.6 in their paper, are evaluated. In our

implementation, we rank all s_2 coefficients from highest to lowest correlation and apply part 2 on all s_2 coefficients in turn according to their ranking. This increases the likelihood of the correct s_2 coefficients being evaluated earlier and eliminates the risk of the correct s_2 coefficient having a correlation below the threshold.

4.5.2 Secret key recovery

After a successful CPA attack, we have obtained all coefficients of the base polynomials in the NTT representation of the secret key. As the secret key in CRYSTALS-Kyber is the byte string encoded NTT representation of the secret key polynomial, we can obtain the secret key by performing the encoding operation on the NTT representation of the secret key, as specified in the key generation procedure in section 2.2.4.

Chapter 5 Results

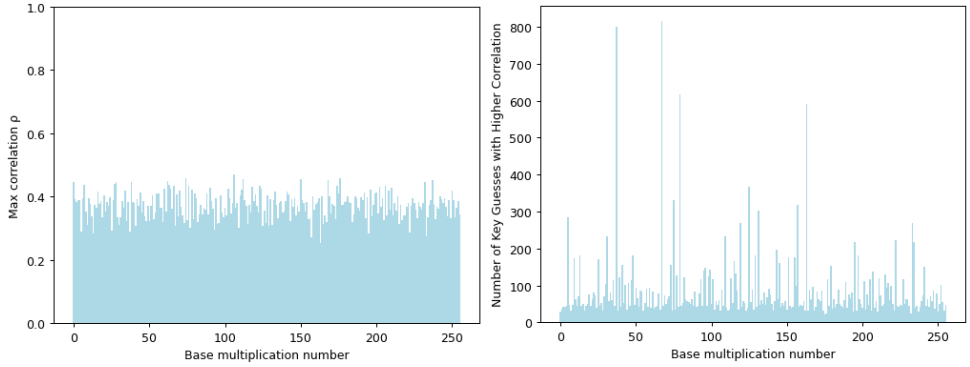
This chapter presents our findings derived from the methodologies described in Chapter 4. We conducted the attack three times, each with different decryption power traces (1000, 200, and 50) against the Kyber512 implementation. The chapter begins by presenting and comparing the initial results from part 1 of the attack. This is followed by the final outcomes after the execution of part 2 of the attack. Next, we compare the runtime of the various attacks. Subsequent sections delve into the limitations of the attack, along with any deviations of the results from the theoretical expectations. Finally, the chapter explores how the attack can be applied to different versions of CRYSTALS-Kyber and CRYSTALS-Dilithium.

5.1 CPA attack part 1

In part 1 of the attack, we rank the correlation for the first step in the NTT domain multiplications of the ciphertext polynomial $(c_1 + c_2X)$ and secret key polynomial $(s_1 + s_2X)$. This is the multiplication $(c_2 \cdot s_2)$, shown in the "smultt" operation in the first line of each "basemul" in Algorithm 4.2. We calculate Hamming distance of the product using all valid s_2 coefficients and rank them according to the maximum correlation with the power usage at a sample point. Part 1 of the attack was performed against all base multiplications, with 1000, 200, and 50 power traces.

5.1.1 1000 traces

In part 1 of the attack using the power traces of 1000 known ciphertext decryptions, the correct key coefficient had an average correlation of 0.37, with each guess having an average of 87.16 coefficients with a higher correlation. Figure 5.1 shows this relation for all base multiplication, with the multiplications in the first dimension $k = 0$ shown in the first half and $k = 1$ shown in the second half. Figure 5.1a shows the correlation of the correct secret key coefficient for all base multiplications, and Figure 5.1b shows the number of key coefficient guesses with a higher correlation than the correct key coefficient for each base multiplication.



(a) **(b)**

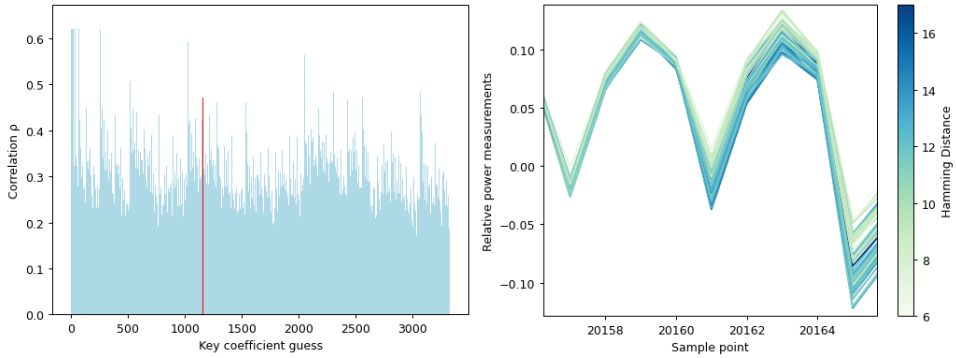
Figure 5.1: For all base multiplications using 1000 power traces, **(a)** shows the correlation for the correct key coefficient and the corresponding rank of each guess is shown in **(b)**.

The base multiplication number 106 in the $k = 0$ dimension had the highest correlation of 0.47, which resulted in having 31 key coefficient guesses with a higher correlation. Figure 5.2a shows the correlation of all key coefficient guesses of this base multiplication, with the correct key coefficient being shown in red. Figure 5.2b shows the raw power trace of each decryption at the sample point where the correct key coefficient achieved the highest correlation (sample point 20161). Each trace is colour-coded with the resulting Hamming distance of the first step of the base multiplication using the correct key coefficient guess, with a high Hamming distance in dark blue and a low Hamming distance in light green.

The correct key coefficient with the worst correlation is found in the base multiplication number 35 in the $k = 1$ dimension with a correlation of 0.25, and having 591 key coefficient guesses with a higher correlation. The correlations of the key coefficient guesses of this base multiplication are found in Figure 5.3a, with the correct key coefficient shown in red. Figure 5.3b shows the power traces at the sampling point where the correct key has the highest correlation (sample point 6741), colour encoded with the corresponding Hamming distance. Compared to the more linear colour shift of Figure 5.2b, the lower power to hamming distance correlation is clearly shown in Figure 5.3b having power traces with high correlation in the middle.

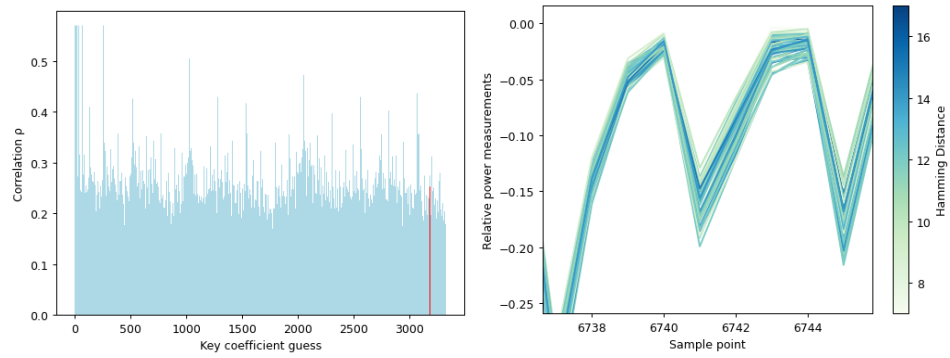
5.1.2 200 traces

This section assesses part 1 of the CPA attack, using 200 power traces of known ciphertext decryptions. The average correlation of the correct key coefficients of all base multiplications is 0.39, with an average of 282.6 key coefficient guesses having



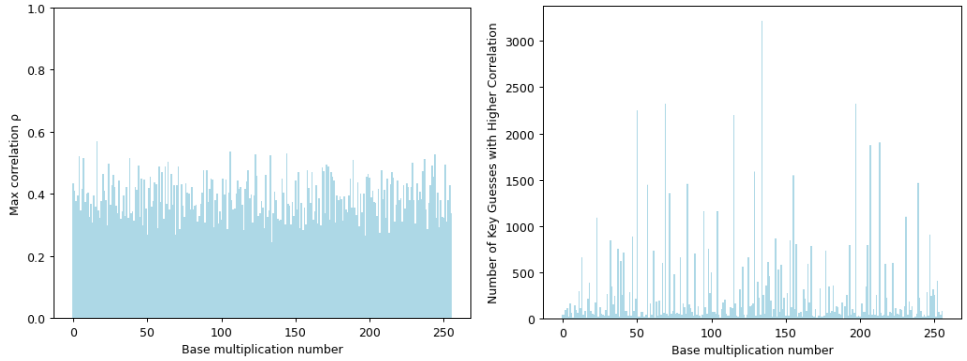
(a) **(b)**

Figure 5.2: Using 1000 power traces, **(a)** shows the correlation of key coefficient guesses of the base multiplication with the highest correlation of the correct key coefficient (shown in red). **(b)** shows the relative power measurements during decryption at the sample points around the maximum correlation (sample point 20161), for the correct key coefficient guess, colour-coded with the Hamming distance after part 1 of the attack (corresponding to the number of bits changed on the databus). Each line represents the power trace of one decryption.



(a) **(b)**

Figure 5.3: **(a)** shows the correlation of the base multiplication with the lowest correlation of the correct key coefficient using 1000 power traces. **(b)** shows the sampling point with the highest correlation for the correct key of this base multiplication, colour coded with Hamming Distance.



(a) **(b)**

Figure 5.4: Correlation **(a)** and rank **(b)** of the correct key coefficient for all base multiplications using 200 traces.

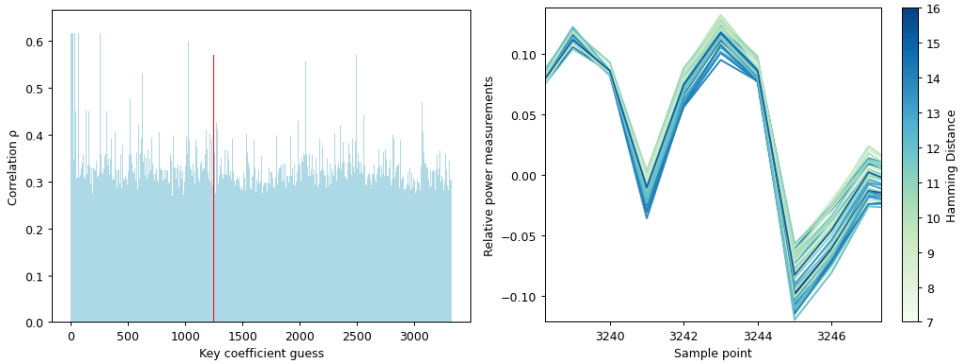
a higher correlation. Figure 5.4a shows the correlation of all base multiplications using the correct s_2 coefficient, while Figure 5.4b shows the amount of incorrect s_2 coefficients with a higher correlation.

The base multiplication with the highest correct key coefficient correlation is the 16th multiplication of the $k = 0$ dimension, resulting in a correlation of 0.57 with 17 key correlation guesses having a higher correlation. Figure 5.5a shows the correlation of all key guesses of this base multiplication with the correct key coefficient shown in red, and Figure 5.5b shows the power traces at the sample point where the correct key coefficient had the best correlation (sample point 3243) colour coded with the Hamming distance.

The worst correct key coefficient correlation is in 6th multiplication in the $k = 1$ dimension with a correlation of 0.24 with 3218 key coefficient guesses having a higher correlation out of 3329 key coefficients, as seen in Figure 5.6a. As seen in Figure 5.6b, there is little correlation between the Hamming distance and the power usage, with the high Hamming distance traces gathering in the middle.

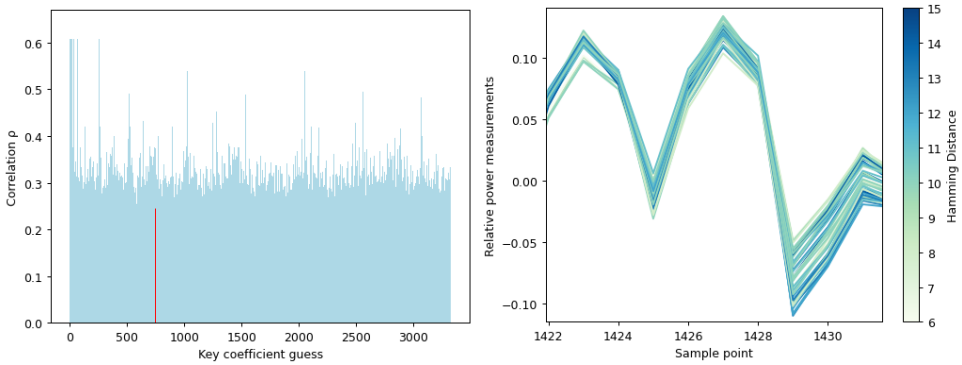
5.1.3 50 traces

For the implementation of the countermeasure suggested by Karlov et al. [KdG21], the correct key coefficients have an average correlation of 0.55 with an average of 1400.8 key coefficient guesses having a higher correlation than the correct key coefficient, as shown in Figure 5.7. An average rank of 1400.8 is only marginally better than randomly ordering the key coefficient, which would result in an average rank of $\frac{q-1}{2} = \frac{3328}{2} = 1664$. However, as part 1 of the attack calculates correlation



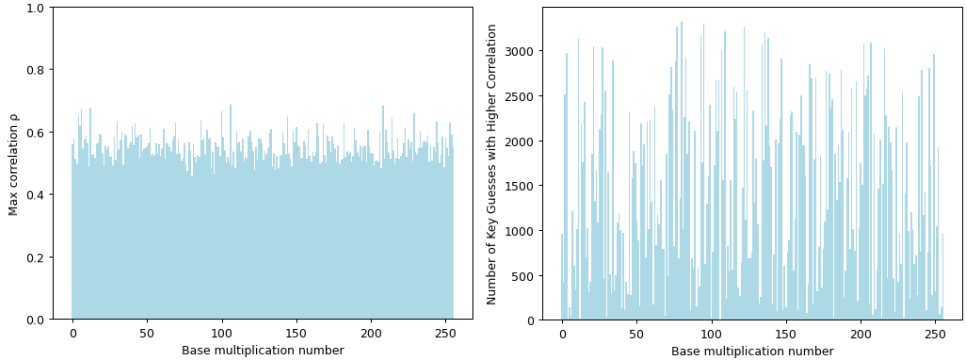
(a) **(b)**

Figure 5.5: Using 200 power traces, **(a)** shows the correlation of the base multiplication with the highest correlation of the correct key coefficient. **(b)** shows the power traces at the sample point where the correct key coefficient had the highest correlation, colour coded with the Hamming distance.



(a) **(b)**

Figure 5.6: **(a)** shows the correlation of key coefficient guesses in the base multiplication where the correct (red) key coefficient have the worst correlation. **(b)** shows the power traces encoded with hamming distance at the sample point (1427) where the correct key coefficient had the highest correlation.



(a) **(b)**

Figure 5.7: The correlation of the correct key coefficient is shown in **(a)**, with the rank of each correct key coefficient shown in **(b)**, for all base multiplications using 50 power traces.

for each key coefficient once for each base multiplication and saves on average $1664 - 1401 = 263$ iterations per base multiplication in part 2 of the attack, part 1 of the attack with 50 power traces is significantly more efficient than performing part 2 of the attack with a random ordering of the s_2 key coefficient.

The best correlation is in the 106th base multiplication in the first ($k = 0$) dimension, as it was with 1000 traces. Figure 5.8a shows the correlation of each key coefficient guess of this base multiplication with the correct key coefficient 1155 shown in red. The correct key coefficient has a correlation of 0.69, which is the 11th highest correlation for this base multiplication. Figure 5.8b shows the power traces colour encoded with the Hamming distance in the sample point where the correct key coefficient had the highest correlation.

The base multiplication with the worst correlation of the correct key coefficient is the 80th base multiplication in the $k = 0$ dimension, shown in Figure 5.9a. The correct key coefficient (shown in red) has a correlation of 0.46 and a rank of 3319, with only 9 key coefficient guesses having a worse correlation. Figure 5.9b shows this lack of correlation has little to no visual correlation between the power usage and Hamming distance in the sample point with the highest correlation of the correct key (sample point 11808).

5.1.4 Summary

The attack is summarised in Table 5.1.

A clear trend emerges that with fewer power traces comes a higher average

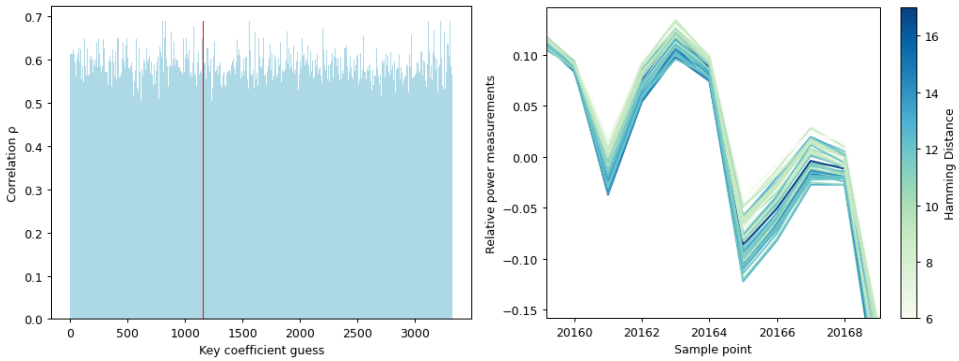


Figure 5.8: **(a)** shows the correlation of key coefficient guesses for the base multiplication where the correct key coefficient had the highest correlation. **(b)** shows the relative power measurements during decryption at the sample point where the correct key coefficient had the maximum correlation (sample point 20164), colour-coded with the Hamming distance.

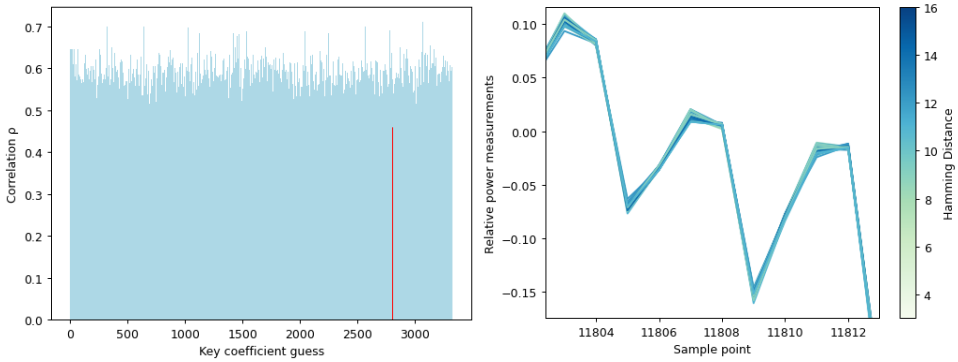


Figure 5.9: The correlation of key coefficient guesses of the polynomial with the lowest correlation of the correct key (shown in red) is shown in **(a)**, with **(b)** showing the power trace of the correct key coefficient at max correlation, colour encoded with the Hamming distance.

Attack Type	Best Case		Worst Case		Average Case	
	Correlation	Rank	Correlation	Rank	Correlation	Rank
1000 Traces	0.47	31	0.25	591	0.37	87.1
200 Traces	0.57	17	0.24	3218	0.39	282.6
50 Traces	0.69	10	0.46	3319	0.55	1400.8

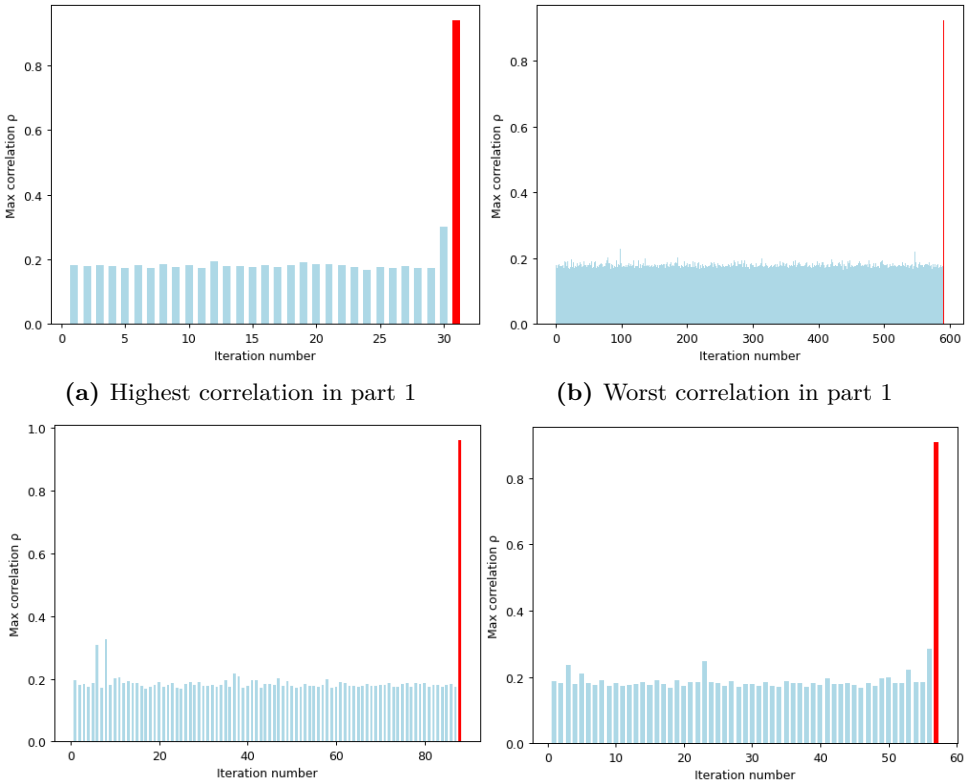
Table 5.1: Comparison of the attack types through best, worst, and average cases of correlation and rank.

correlation and a higher amount of key coefficient guesses with a higher correlation than the correct key. This is expected, as with more power traces, there are less likely to be random points with high correlation, making the correct key coefficient guess more likely to stand out. This effect is observable in the worst and average case of each attack type, with the worst correct correlation with 1000 power traces still being in the top 20% of the key guesses, while the worst case for 200 and 50 power traces is in the bottom 5%.

5.2 CPA attack part 2

Part 2 of the attack is also done one base multiplication at a time. The attack iterates through all s_2 coefficients ranked in part 1 from highest to lowest correlation and calculates the Hamming distance of the whole base multiplication $((c_1 + c_2X) \cdot (s_1 + s_2X))$ for all possible values of s_1. The resulting polynomial is stored in memory in the last line of each "basemul" in Algorithm 4.2. Then the correlation between the Hamming distance and the power usage of each ciphertext is calculated. If a correlation is higher than a fixed value, the attack assumes that the correct secret coefficients, s_1 and s_2, are found.

Because of the long runtime of the attack, the full part 2 of the attack was run against a selection of the base multiplications. For all attack types, the full attack was performed against the base multiplications with the highest and lowest correlation in part 1 of the attack and against base multiplications with the highest and lowest correlation in part 2 of the attack. The attack was performed for the rest of the base multiplications only using the correct s_2 coefficients from part 1. Ensuring the attack will detect the correct key in all base multiplications and obtaining useful data as the maximum, minimum and average correlation of the correct key guesses. However, there is not known how many, if any, false positives will be detected for the base multiplications where the full attack was not performed.



(a) Highest correlation in part 1

(b) Worst correlation in part 1

(c) Highest correlation in part 2

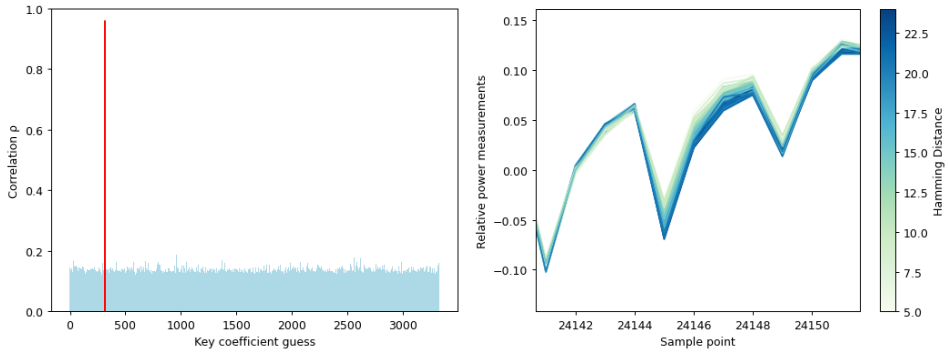
(d) Worst correlation in part 2

Figure 5.10: Maximum correlation in all iterations where the full part 2 of the attack was performed. The iterations using the correct part 1 key coefficient (s_2) is shown in red

5.2.1 1000 traces

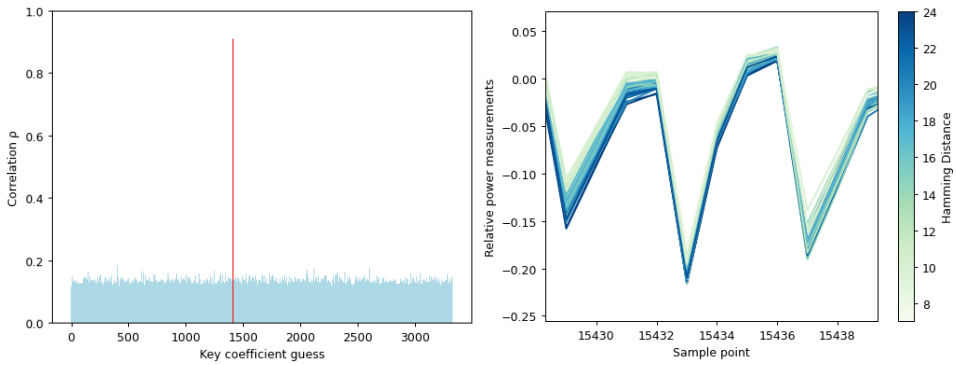
In the attack using 1000 power traces, the threshold correlation for finding the key was set to 0.9. Using the correct key coefficients resulted in an average correlation of 0.93 with a maximum value of 0.96 and a minimum value of 0.91. Figure 5.10 shows the correlation of all iterations in all base multiplications where the full attack was performed. The figure clearly shows only the iterations using the correct s_2 key coefficient (shown in red) come close to reaching the threshold value of 0.9. Making it unlikely any false s_2 coefficients would be obtained by running the full attack against all base multiplications.

Figure 5.11a and Figure 5.12a show the correlation of all s_1 key coefficient guesses using the correct value of s_2 for the base multiplication with the best and worst



(a) (b)

Figure 5.11: The base multiplication where the correct key coefficient had the best correlation in part 2 of the attack using 1000 traces.



(a) (b)

Figure 5.12: The base multiplication where the correct key coefficient had the worst correlation in part 2 of the attack using 1000 traces.

correlation respectively, with the correct value of s_1 being shown in red. As no other values of s_1 are close to reaching the threshold of 0.9, the attack with 1000 power traces is unlikely to result in a false positive for s_1. Figure 5.11b and Figure 5.12b show the power trace of each decryption at the sample point with the highest correlation colour coded with the Hamming distance obtained using the correct key coefficients. The gradual colour change of these figures clearly shows the linear relationship between power usage and Hamming distance.

5.2.2 200 traces

The threshold correlation for finding the key was also set to 0.9 in the attack using 200 power traces. Using the correct key coefficients, an average correlation of 0.93, a maximum of 0.96 and a minimum of 0.91 was obtained. Figure 5.13 shows the max correlation of all iterations through s_2 from part 1 of the attack for all the base multiplications where the full attack was executed. For the iterations using the incorrect s_2 key coefficient, a majority of the correlation is around 0.4. Some notable exceptions are found in Figure 5.13b where the 3146th iteration has a correlation of 0.81, nearly passing the threshold of 0.9, which would result in a false positive. Therefore a full attack against all base multiplication may have resulted in a few erroneous key coefficients.

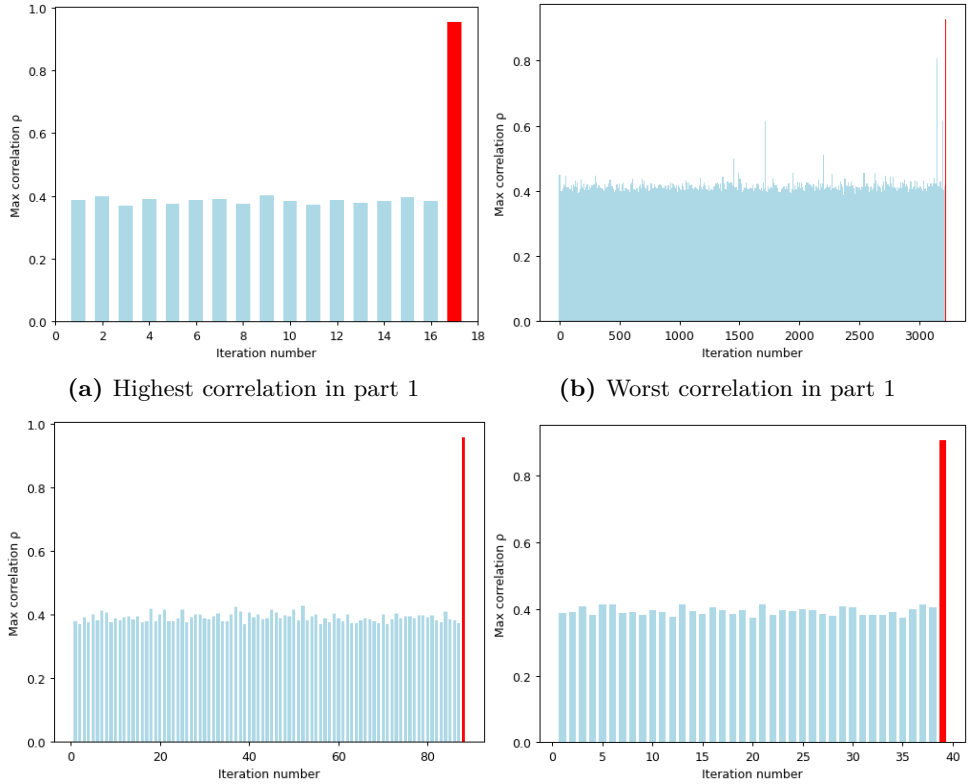
The attack would still be successful with a few false positives. As all correct key coefficients stand out with a correlation at or above 0.91, the correct key coefficients would be found by continuing to iterate through the s_2 coefficients until another coefficient with a high correlation is found. However, as it would not be known what base multiplication resulted in an erroneous key coefficient, all base multiplications must be searched.

The correlation of the s_1 coefficient guesses using the correct s_2 coefficient is shown in Figure 5.14a for the base multiplication which resulted in the highest correlation, and in Figure 5.15a for the base multiplication which resulted in the worst correlation. The correct s_1 coefficient is shown in red. The incorrect s_1 coefficients have a correlation of around 0.3 and are unlikely to result in an erroneous key coefficient guess. However, if they do, the attack should still be successful by following the procedure described in the previous paragraph. Figure 5.14b and Figure 5.15b show the linear relationship between the Hamming distance and power usage by plotting the power traces colour coded with the Hamming distance at the sample point where the correct key coefficient had the highest correlation.

5.2.3 50 traces

This section shows the results of the attack using 50 power traces. As fewer traces introduce more noise, it was decided to reduce the correlation threshold for finding the key coefficients to 0.85. The attack resulted in the correct key coefficients achieving an average correlation of 0.94, a maximum of 0.97 and a minimum of 0.87.

The full attack was performed against four base multiplications, which resulted in one false positive and three correct key coefficient pairs. The attack was performed against the base multiplications with the best and worst correlation in both parts of the attack. The best correlation for all iterations of the s_2 key coefficient from part



(a) Highest correlation in part 1

(b) Worst correlation in part 1

(c) Highest correlation in part 2

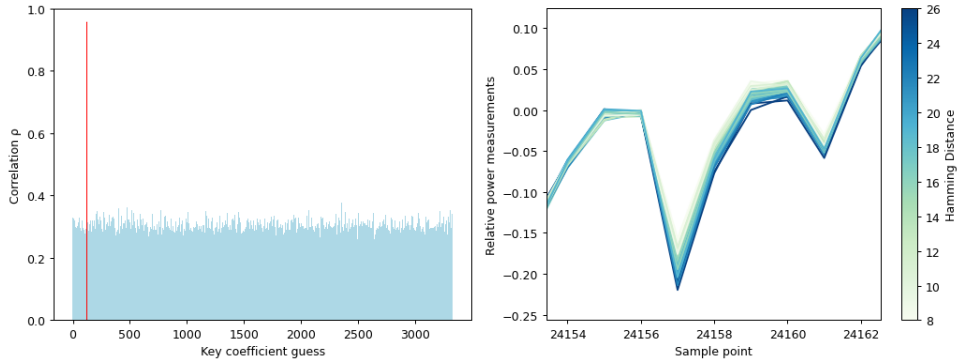
(d) Worst correlation in part 2

Figure 5.13: Maximum correlation in all iterations where the full part 2 of the attack was performed using 200 power traces. The iterations using the correct part 1 key coefficient (s_2) are shown in red

1 of the attack is shown in Figure 5.16, with the max correlation of the correct s_2 coefficient shown in red.

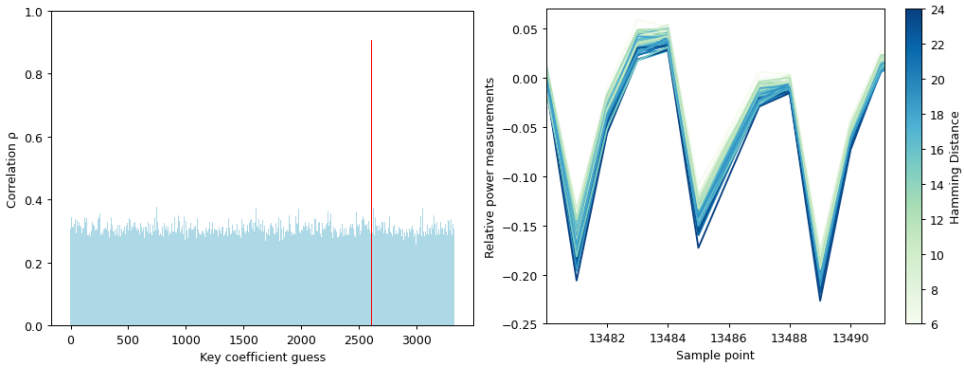
The false positive was obtained in the base multiplication with the highest correct key coefficient in part 2, shown in Figure 5.16c. In the 2367th iteration, incorrect key coefficients obtained a correlation of 0.90, much higher than the threshold of 0.85 and the correct key coefficients with the lowest correlation of 0.87. As shown in Figure 5.16c and Figure 5.17a the correct key coefficients were still obtained by continuing to iterate through the s_2 coefficients until another high correlation combination was found.

Figure 5.16 also shows that the correct s_2 coefficients stand out from the majority of the incorrect s_2 coefficients. And since the lowest correct key coefficients have



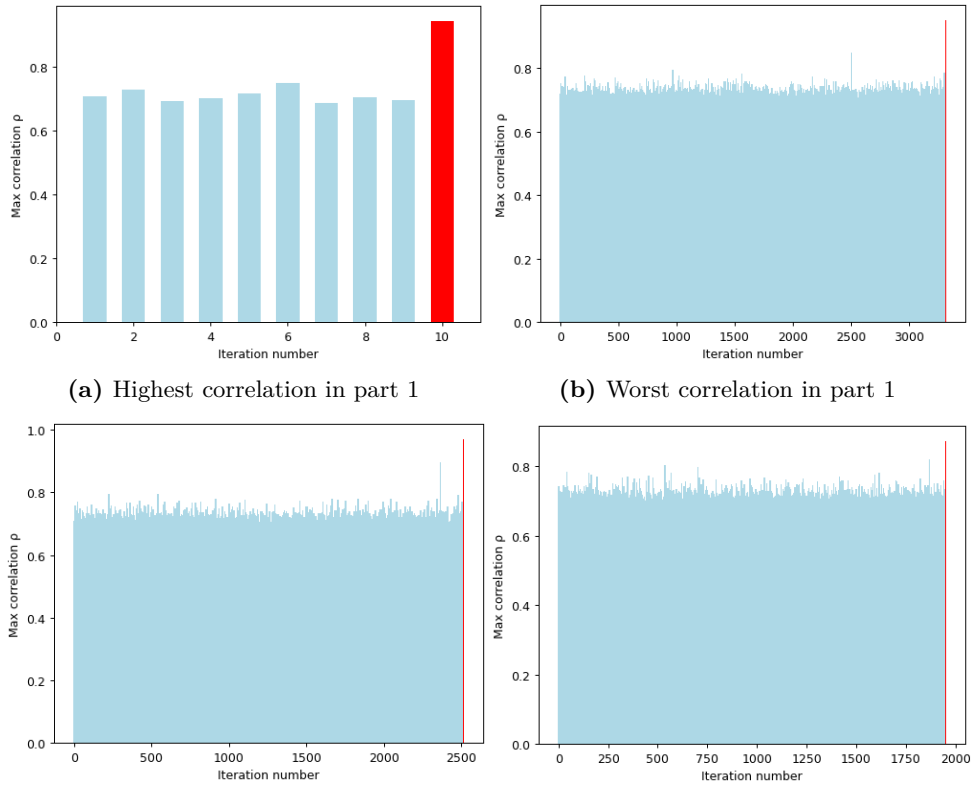
(a) **(b)**

Figure 5.14: The base multiplication where the correct key coefficient had the best correlation in part 2 of the attack using 200 traces.



(a) **(b)**

Figure 5.15: The base multiplication where the correct key coefficient had the worst correlation in part 2 of the attack using 200 traces.



(a) Highest correlation in part 1

(b) Worst correlation in part 1

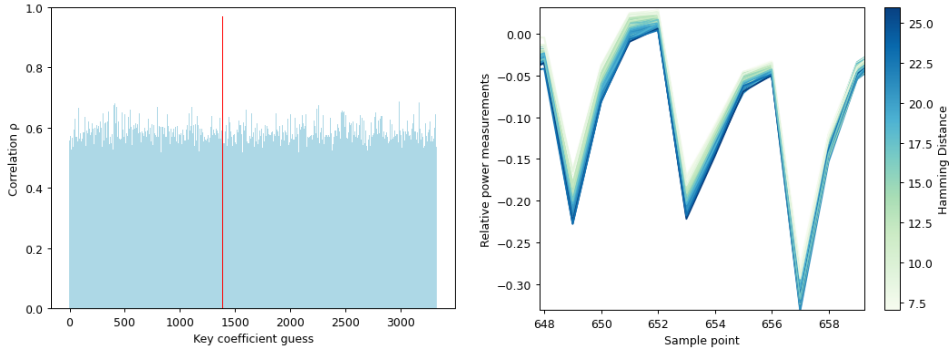
(c) Highest correlation in part 2

(d) Worst correlation in part 2

Figure 5.16: Maximum correlation in all iterations where the full part 2 of the attack was performed using 50 power traces. The iterations using the correct part 1 key coefficient (s_2) are shown in red

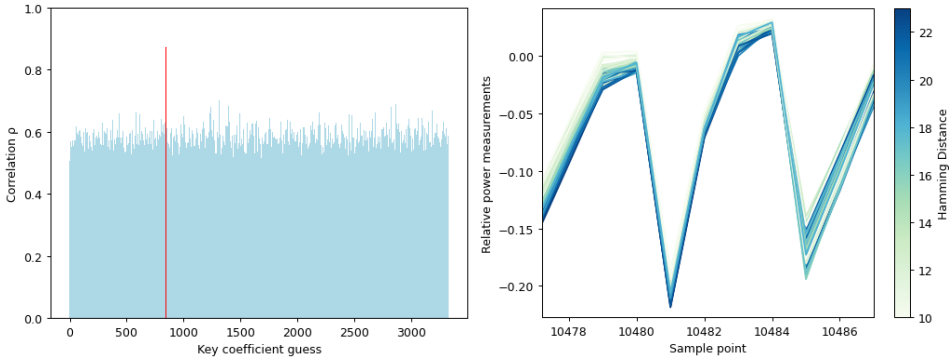
a correlation of 0.87, all correct s_2 coefficients will eventually be found using the above method of continuing to the search until the next coefficients with a high correlation. Figure 5.17 and Figure 5.18 also show the correct combination of the s_2 and s_1 coefficients clearly stand out from the correlation of the correct s_2 and incorrect s_1 coefficient.

By successfully recovering the secret key coefficients in four out of four base multiplications, we conclude that the countermeasure introduced by Karlov et al. [KdG21] of changing the secret key after every 50 communication cycle is insufficient for preventing a secret key recovery as it is possible to obtain the 50 power traces used in the attack with the countermeasure implemented. Given that the correlation of correct key coefficients still differs from the incorrect ones, the attack is likely feasible with even fewer power traces.



(a) **(b)**

Figure 5.17: The base multiplication where the correct key coefficient had the best correlation in part 2 of the attack using 50 traces.



(a) **(b)**

Figure 5.18: The base multiplication where the correct key coefficient had the worst correlation in part 2 of the attack using 50 traces.

5.2.4 Summary

The secret key coefficients were recovered in all base multiplications using 1000, 200, and 50 power traces. Table 5.2 summarises our findings from part 2 of the attack. The correlation using the correct key coefficients is similar in all attack types, with a slightly bigger variance in the attack using 50 traces. The main difference between the attack types is the average correlation when using incorrect key coefficients, with fewer power traces resulting in higher correlation, which again increases the likelihood of false positives.

Attack Type	Best (c)	Worst (c)	Average (c)	Average (i)
1000 Traces	0.96	0.91	0.93	0.17
200 Traces	0.96	0.91	0.93	0.37
50 Traces	0.97	0.87	0.94	0.73

Table 5.2: Comparison of attack types through maximum, worst and average correlation, differentiating correct (c) and incorrect (i) key coefficients. Note that the averages using the incorrect key coefficients are approximate, as the full attack was only performed against four base multiplication for each attack type.

5.3 Attack runtime

Two main factors dictate the attack's run time: the number of iterations required for finding all key coefficients and the speed of performing one iteration. By iteration, we mean iterating through all values of one key coefficient s_1 or s_2.

The amount of iterations required largely depends on the rank of the correct key coefficient in part 1 of the attack. In one base multiplication, the following minimum amount of iterations is required. Additional iterations are likely needed in case of false positives.

- One iteration in part 1, iterating through and ranking all s_2 coefficients.
- One iteration for all s_2 coefficients with a higher rank than the correct s_2 coefficient, with each iteration iterating through all s_1 coefficients.
- One iteration for the correct s_2 coefficient, iterating until the correct s_1 coefficient is found.

The speed of one iteration largely depends on the size of the data being compared, the amount of data points in the set of Hamming distances and the set of power measurements used in each correlation equation, and the number of correlation equations required in the form of sample points. In all attack types, 24400 sample points were used. For the attack using 1000 power traces, each correlation equation uses 1000 Hamming distances and 1000 power measurement points. The attack using 200 power traces calculates correlation using 200 Hamming distances and 200 power measurement points. And lastly, in the attack using 50 power traces, each correlation equation uses 50 Hamming distances and 50 power measurement points.

The results are shown in Table 5.3. Note that the time usage was recorded using an Intel Core i7-13700KF processor, and the average and total time usage was

calculated using the average time per iteration in the four base multiplications used in the full attack. The total time usage also assumes no false positives were found.

Attack Type	Best		Worst		Average		Total	
	i	t(s)	i	t(s)	i	t(s)	i	t(s)
1000 Traces	33	5856	593	105580	87.2	15502	22825	4059565
200 Traces	19	680	3220	116633	282.6	10187	72847	2625686
50 Traces	12	109	3321	30694	1402.8	12853	359118	3295066

Table 5.3: Comparison of attack types through time usage in seconds t(s) and the number of iterations (i) required.

Table 5.3 shows that the decreased number of iterations required to perform the attack with 1000 power traces is insufficient to account for the heavy calculations needed for each iteration, making the full attack take 47 days compared to the 30 days required in the attack using 200 power traces. The opposite is the case for the attack using 50 traces, where the decreased computation time is insufficient to account for the high number of iterations required, making the attack take 38 days to be completed. However, the true time usage is expected to be longer due to the high number of false positives expected in the attack using 50 traces. In a worst-case scenario where the true s_2 coefficient has the worst correlation in all base multiplications, the true key coefficients are still found within $\frac{3295066s}{1402.8i} \cdot 3329i \approx 7819557s \approx 91$ days after iterating through all values of s_2 in all base multiplications.

Some optimizations for the attack are possible, most notably in limiting the number of sample points to be searched for a high correlation. In our implementation, all sample points are searched in all base multiplications. As the Kyber512 implementation sequentially calculates the base multiplication, no base multiplication happens before the sample point where the previous base multiplication had the maximum correlation. Similarly, all base multiplication occurs shortly after each other due to close code proximity. Therefore a higher efficiency could be achieved by, for example, only searching the 1000 sample points directly after the sample point where the key coefficients of the previous base multiplication were found. In testing, this approach had little effect on the time used in each iteration. Still, it drastically decreased the number of incorrect key coefficients with a higher correlation than the correct one in part 1 of the attack, reducing the number of iterations required in part 2. However, this approach may result in consequential errors if false positive key coefficients were obtained in the previous base multiplication.

5.4 Limitations

Part 1 of the attack does not return valid results for base multiplications where the s_2 secret coefficient is 0. As part 1 targets the multiplication $s_2 \cdot c_2$, the product and Hamming distance using $s_2 = 0$ will be zero for all ciphertext c_2 coefficient values, which does not leak any information.

Our implementation does not consider this, resulting in $s_2 = 0$ having the highest correlation in part 1 in all base multiplications. When all Hamming distances are zero, the set has zero variance ($\sigma_H = 0$), resulting in the Pearson correlation coefficient converging towards infinity ($\rho_{H,P} = \frac{cov(H,P)}{\sigma_H \sigma_P}$).

5.5 Diversions from theory

According to theory, a higher Hamming distance should result in higher power usage. However, the results (as shown in Figure 5.11b) show that a higher Hamming distance results in lower power usage. This is due to the measurement setup in the ChipWhisperer Lite. The ChipWhisperer obtains power usage by measuring voltage drop on one side of the shunt resistor. This results in a higher voltage when no current flows through the resistor, inverting the result [NewAEb].

5.6 Applying the attack to other versions of CRYSTALS-Kyber

The main difference between Kyber512, Kyber768 and Kyber1024 is the number of dimensions in the form of k, with each version having $k = 2$, $k = 3$ and $k = 4$ respectively. As we attack one dimension at a time, an increase in k will result in a linear increase in attack runtime. With an attack against Kyber768 taking 50% longer and Kyber1024 taking double the amount of time compared to our attack against Kyber512.

5.7 Applying the attack to CRYSTALS-Dilithium

In CRYSTALS-Dilithium, the NTT representation of polynomials are 0-degree polynomials or constants [DKL+21]. However, the modulus q is set to 8380417 [DKL+21], requiring the correlation of 8380417 constants to be evaluated for each base multiplication in an attack. This will likely make the attack less efficient than the attack against CRYSTALS-Kyber, as there is not possible to split the attack into two shorter coefficients with 3329 possible values each. However, the attack should be more efficient than brute forcing both Kyber coefficients with $3329^2 = 11082241$ possibilities.

Compared to CRYSTALS-Kyber, CRYSTALS-Dilithiums secret key consists of 6 elements, ρ, K, tr, and the polynomials $\mathbf{s}_1$, $\mathbf{s}_2$ and $\mathbf{t}_0$. Both ρ and tr are publicly available as ρ is a part of the public key, and tr is the public key hash. K is the preimage under a hash function and is not involved in a polynomial multiplication, making a full secret key recovery against CRYSTALS-Dilithium impossible with our attack. However, the secret key parameters $\mathbf{s}_1$, $\mathbf{s}_2$ and $\mathbf{t}_0$ are involved in the NTT domain multiplications $\hat{c} \cdot \hat{\mathbf{s}}_1$, $\hat{c} \cdot \hat{\mathbf{s}}_2$ and $\hat{c} \cdot \hat{\mathbf{t}}_0$ as part of the signing algorithm[DKL+21]. The polynomial $\hat{c}$ is publicly available from the public key parameter $\tilde{\mathrm{c}}$, making $\mathbf{s}_1$, $\mathbf{s}_2$ and $\mathbf{t}_0$ vulnerable for recovery using our attack.

This attack is likely difficult to implement using the hardware used in this thesis due to memory limitations. We had memory issues in implementing Kyber512 as a KEM, and CRYSTALS-Dilithium will likely require more memory due to increased key and signature sizes [DKL+21].

Chapter 6
Conclusion

The primary aim of this thesis was to assess the security strength of CRYSTALS-Kyber, focusing on the power analytic side-channel attack and its possible enhancements and countermeasures. We conclude by looking back at the research questions.

Can the correlational power analytical attack performed by Karlov et al. be replicated?

Our method for recreating the power analytical SCA by Karlov et al. [KdG21] is provided in Chapter 4. The technique successfully recovered the partial key against Kyber512 in all our attacks, demonstrating that this approach can recover the full secret key effectively. Our analysis also found the attack to apply against all versions of CRYSTALS-Kyber, with a linear increase in time as the security parameter increases.

Can the attack be improved?

Our modifications to the original attack included limiting the number of coefficients required for testing, adhering strictly to the Kyber specifications, and avoiding the threshold requirement in part 1 of the attack, described in Chapter 4. These adjustments improved the attack's efficiency and ensured correct coefficients were evaluated in part 2, even with low correlation.

How does the availability of varying amounts of power traces impact the performance of the attack?

In our assessment of the attack's efficiency (in Chapter 5), we discovered more traces drastically reduced the risk of false positives and the number of calculations required. Interestingly, due to each calculation's size, the attack's runtime increased drastically with a high amount of traces. Requiring future implementations consider the tradeoff between the runtime and accuracy of the attack.

Can the countermeasures introduced by Karlov et al. effectively mitigate the attack?

A critical outcome of our research is the demonstration that the countermeasure suggested by Karlov et al. [KdG21] was ineffective. We implemented the countermeasure of limiting the attack to 50 power traces in section 5.2.3 and recovered the secret key coefficients in all attempted base multiplications using our improved attack.

Is the attack applicable to the corresponding signature scheme, CRYSTALS-Dilithium?

We are quite confident our attack partially applies against CRYSTALS-Dilithium, with it likely being possible to recover all parts of the secret keys involved in a base multiplication as discussed in section 5.7. However, full secret key recovery is beyond the scope of our attack as we leave it for future works to determine ways to recover the secret key of CRYSTALS-Dilithium fully.

References

[07] «Power Consumption», in *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Boston, MA: Springer US, 2007, pp. 27–60. [Online]. Available: https://doi.org/10.1007/978-0-387-38162-6_3.

[ABL+21] R. Avanzi, J. Bos, *et al.*, *CRYSTALS-Kyber algorithm specifications and supporting documentation*, version 3.02, Aug. 2021.

[Ati18] M. Atiyah, *Introduction to commutative algebra*. CRC Press, 2018.

[BCNS14] J. W. Bos, C. Costello, *et al.*, *Post-quantum key exchange for the tls protocol from the ring learning with errors problem*, Cryptology ePrint Archive, Paper 2014/599, https://eprint.iacr.org/2014/599, 2014. [Online]. Available: https://eprint.iacr.org/2014/599.

[BCO04] E. Brier, C. Clavier, and F. Olivier, «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*, Springer, 2004, pp. 16–29.

[BGV14] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, «(leveled) fully homomorphic encryption without bootstrapping», *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.

[BHLY16] L. G. Bruinderink, A. Hülsing, *et al.*, *Flush, gauss, and reload – a cache attack on the bliss lattice-based signature scheme*, Cryptology ePrint Archive, Paper 2016/300, https://eprint.iacr.org/2016/300, 2016. [Online]. Available: https://eprint.iacr.org/2016/300.

[Cor99] J.-S. Coron, «Resistance against differential power analysis for elliptic curve cryptosystems», in *Cryptographic Hardware and Embedded Systems: First InternationalWorkshop, CHES'99 Worcester, MA, USA, August 12–13, 1999 Proceedings 1*, Springer, 1999, pp. 292–302.

[DKL+21] L. Ducas, E. Kiltz, *et al.*, «CRYSTALS-Dilithium – algorithm specifications and supporting documentation», Tech. Rep., version 3.1, Feb. 2021. [Online]. Available: https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf.

[FO99] E. Fujisaki and T. Okamoto, «Secure integration of asymmetric and symmetric encryption schemes», in *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings*, Springer, 1999, pp. 537–554.

[GT03] J. D. Golić and C. Tymen, «Multiplicative masking and power analysis of AES», in *Cryptographic Hardware and Embedded Systems-CHES 2002: 4th International Workshop Redwood Shores, CA, USA, August 13–15, 2002 Revised Papers 4*, Springer, 2003, pp. 198–212.

[IBMQuantum] Real quantum computers. [Online]. Available: https://quantum-computing.ibm.com/ (last visited: Jun. 3, 2023).

[KdG21] A. Karlov and N. L. de Guertechin, «Power analysis attack on Kyber», *Cryptology ePrint Archive*, 2021.

[KJJ99] P. Kocher, J. Jaffe, and B. Jun, «Differential power analysis», in *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, Springer, 1999, pp. 388–397.

[KL20] J. Katz and Y. Lindell, *Introduction to modern cryptography*. CRC press, 2020.

[KPR+] M. J. Kannwischer, R. Petri, *et al.*, *PQM4: Post-quantum crypto library for the ARM Cortex-M4*, https://github.com/mupq/pqm4.

[LPR13] V. Lyubashevsky, C. Peikert, and O. Regev, «On ideal lattices and learning with errors over rings», *Journal of the ACM (JACM)*, vol. 60, no. 6, pp. 1–35, 2013.

[LS15] A. Langlois and D. Stehlé, «Worst-case to average-case reductions for module lattices», *Designs, Codes and Cryptography*, vol. 75, no. 3, pp. 565–599, 2015.

[Lyu20] V. Lyubashevsky, «Basic lattice cryptography: Encryption and fiat-shamir signatures», IBM Research - Zurich, Säumerstrasse 4, 8803 Rüschlikon, Switzerland, Tech. Rep., Dec. 2020, Original Version: December 2019. Last Updated December 17, 2020.

[MMS01] D. May, H. L. Muller, and N. P. Smart, «Non-deterministic processors», in *Information Security and Privacy: 6th Australasian Conference, ACISP 2001 Sydney, Australia, July 11–13, 2001 Proceedings 6*, Springer, 2001, pp. 115–129.

[MNM+16] T. Monz, D. Nigg, *et al.*, «Realization of a scalable Shor algorithm», *Science*, vol. 351, no. 6277, pp. 1068–1070, 2016.

[NewAEa] ChipWhisperer. [Online]. Available: https://chipwhisperer.readthedocs.io/ (last visited: Apr. 12, 2023).

[NewAEb] NewAE Hardware Product Documentation. [Online]. Available: https://rtfm.newae.com/ (last visited: Apr. 11, 2023).

[NIST] Post-Quantum Cryptography Standardization. [Online]. Available: https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization (last visited: Jun. 3, 2023).

[Nov02] R. Novak, «SPA-based adaptive chosen-ciphertext attack on RSA implementation», in *Public Key Cryptography: 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002 Paris, France, February 12–14, 2002 Proceedings*, Springer, 2002, pp. 252–262.

[OC14] C. O'flynn and Z. Chen, «Chipwhisperer: An open-source platform for hardware embedded security research», in *Constructive Side-Channel Analysis and Secure Design: 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers 5*, Springer, 2014, pp. 243–260.

[Reg09] O. Regev, «On lattices, learning with errors, random linear codes, and cryptography», *Journal of the ACM (JACM)*, vol. 56, no. 6, pp. 1–40, 2009.

[Reg10] O. Regev, «The learning with errors problem», *Invited survey in CCC*, vol. 7, no. 30, p. 11, 2010.

[SBB18] P. Socha, J. Brejník, and M. Bartik, «Attacking AES implementations using correlation power analysis on ZYBO Zynq-7000 SoC board», in *2018 7th Mediterranean Conference on Embedded Computing (MECO)*, IEEE, 2018, pp. 1–4.

[Sho94] P. W. Shor, «Algorithms for quantum computation: Discrete logarithms and factoring», in *Proceedings 35th annual symposium on foundations of computer science*, Ieee, 1994, pp. 124–134.

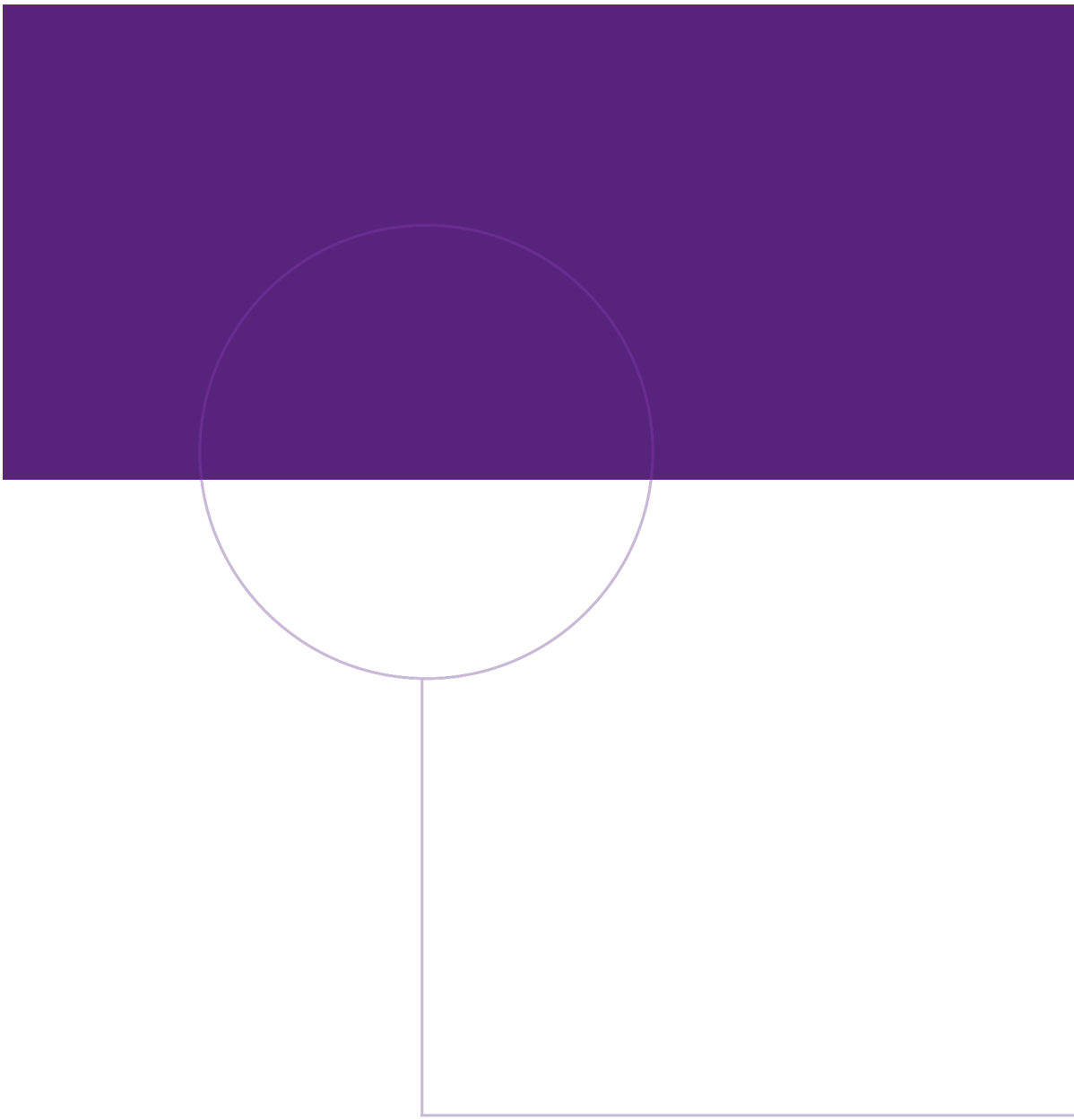
[Sta10] F.-X. Standaert, «Introduction to side-channel attacks», *Secure integrated circuits and systems*, pp. 27–42, 2010.

[WM21] M. E. Whitman and H. J. Mattord, *Principles of information security*. Cengage learning, 2021.

[Wri87] P. Wright, *Spycatcher: The Candid Autobiography of a Senior Intelligence Officer*. Heinemann, 1987.

[ZF05] Y. Zhou and D. Feng, «Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing», *Cryptology ePrint Archive*, 2005.

[ÖOP03] S. B. Örs, E. Oswald, and B. Preneel, «Power-analysis attacks on an FPGA–first experimental results», in *Cryptographic Hardware and Embedded Systems-CHES 2003: 5th International Workshop, Cologne, Germany, September 8–10, 2003. Proceedings 5*, Springer, 2003, pp. 35–50.



NTNU

Norwegian University of
Science and Technology