Decryption Failure Attacks on Post-Quantum Cryptography

Nilsson, Alexander

2023

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
Nilsson, A. (2023). *Decryption Failure Attacks on Post-Quantum Cryptography*. [Doctoral Thesis (compilation), Department of Electrical and Information Technology]. Lunds Universitet/Lunds Tekniska Högskola.

Total number of authors:
1

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Decryption Failure Attacks on Post-Quantum Cryptography

Alexander Nilsson

LUND UNIVERSITY

ISBN 978-91-8039-695-0 (print)
ISBN 978-91-8039-696-7 (electronic)
Series of licentiate and doctoral theses
No. 155
ISSN 1654-790X

Alexander Nilsson
Department of Electrical and Information Technology
Lund University
Box 118
SE-221 00 Lund
Sweden

Typeset using LaTeX.

Cover image illustrating the study of small details in complex systems. Generated by the AI *DALL·E 2* using prompt “pencil drawing of complex old style machine, comical approach”. Edited by Liza Nilsson.

Printed in Sweden by Tryckeriet i E-huset, Lund, 2023.

Compilation no. 9754 based on rev 9d60cd64 from
2023-04-11T23:43:39+02:00

Abstract

This dissertation discusses mainly new cryptanalytical results related to issues of securely implementing the next generation of asymmetric cryptography, or Public-Key Cryptography (PKC).

PKC, as it has been deployed until today, depends heavily on the integer factorization and the discrete logarithm problems. Unfortunately, it has been well-known since the mid-90s, that these mathematical problems can be solved due to Peter Shor's algorithm for quantum computers, which achieves the answers in polynomial time. The recently accelerated pace of R&D towards quantum computers, eventually of sufficient size and power to threaten cryptography, has led the crypto research community towards a major shift of focus.

A project towards standardization of Post-quantum Cryptography (PQC) was launched by the US-based standardization organization, NIST. PQC is the name given to algorithms designed for running on classical hardware/software whilst being resistant to attacks from quantum computers. PQC is well suited for replacing the current asymmetric schemes. A primary motivation for the project is to guide publicly available research toward the singular goal of finding weaknesses in the proposed next generation of PKC.

For public key encryption (PKE) or digital signature (DS) schemes to be considered secure they must be shown to rely heavily on well-known mathematical problems with theoretical proofs of security under established models, such as indistinguishability under chosen ciphertext attack (IND-CCA). Also, they must withstand serious attack attempts by well-renowned cryptographers both concerning theoretical security and the actual software/hardware instantiations. It is well-known that security models, such as IND-CCA, are not designed to capture the intricacies of inner-state leakages. Such leakages are named side-channels, which is currently a major topic of interest in the NIST PQC project.

This dissertation focuses on two things, in general: 1) how does the low but non-zero probability of decryption failures affect the cryptanalysis of these new PQC candidates? And 2) how might side-channel vulnerabilities inadvertently be introduced when going from theory to the practice of software/hardware implementations? Of main concern are PQC algorithms based on lattice theory and coding theory.

The primary contributions are the discovery of novel decryption failure side-channel attacks, improvements on existing attacks, an alternative implementation to a part of a PQC scheme, and some more theoretical cryptanalytical results.

Acknowledgements

I dedicate this thesis to my wife, Sofia, for without you I would not have the strength or courage to even attempt anything so foolish as to follow my childhood dream, and now it's your turn, I love you and support you in this and everything else.

Moving on, I naturally wish to extend my undying gratitude towards the persons of most material guidance, my supervisors and co-authors. Thank you, Thomas, for allowing me to sit on your proverbial giant's shoulders during the writing of what turned out to be all of the included papers. Not to mention, this entire journey. Thank you, Paul, for without your bright ideas, experience, and humor the beginning of my academic career would have failed fast, or at least would have looked unrecognizably different, and I would've been the worse for it. Thank you, Qian, for you are without a doubt the one who, by sheer brilliance, forced me to improve my own faculties the most, just to try and keep up with your next great idea or suggestion, of which there were many.

I further wish to thank my other co-authors, of which there are quite a few. Thank you Boris and Irina for coming to my aid and rescuing an otherwise doomed project, on which I had been stuck for quite a while. I have two merged papers and thus I wish to extend my sincere thanks to my almost accidental co-authors Jan-Pieter D'Anvers, Frederik Vercauteren, Ingrid Verbauwhede, Clemens Hlauschek, Norman Lahr, and Robin Leander Schröder for not only allowing the merges in the first place, but also for being such good sports about the entire endeavor. I would be happy to repeat the process. Thank you, Joakim and Pegah, for working with me on my other contributions, not included in this dissertation. From a social point of view, I will certainly miss our collaboration most of all, you are both great people and super co-workers. Pegah, you deserve a second thank you for additionally being such a great office roommate these last few years, even though most of my work was from home, much due to the pandemic.

Advenica, my first and so far only, permanent position employer holds a special place in my mind. The people there, past and present, deserve all the acknowledgments I can put into words. Thank you, Marie, CFO and CEO, for approving the economics of the situation, it's no small matter and I'm forever grateful. A special thank you goes to Helena who always was the guiding star against whom I measured my own potential future at the company. Without your support, none of this would have happened. Thank you Jonas for spearheading the role of industrial supervisor when Helena was no longer available, and ensuring the support of the company when I needed it. Sebastian, you taking over the supervising role from Jonas deserve a thank you for this of course, but more importantly, you deserve it for being such a nice friend, coworker, discussion board, and let's face it, boss, for some time. I also wish to thank my current boss, Kalle, for always being so understanding of the practicalities of PhD-studies and how it affected my productivity at the company, most probably from having a Ph.D. yourself, though you very rarely bring it up. Håkan, you are undoubtedly the one who saw most of my plights and most ably and kindly explained to me what I needed to hear, by drawing from your own extensive experiences of the academic world and how it applies to the industry. Thank you. There are many many others at the company with whom I have enjoyed and learned much from, too many to list them all. If you feel offended that you are not mentioned by name, I'm sorry but there are so many of you.

Speaking of coworkers, I have not yet mentioned those from the department in Lund who has been instrumental in showing me what it means to be a Ph.D. student. On top of those I have already mentioned as my co-authors, I wish to especially thank Linus, Erik and Jonathan whom I consider to be "the original Ph.D. students". Sometimes leading by example, sometimes acting as a deterrent, you are a gift to humanity. Linus, I'm proud and happy that our always interesting and occasionally deep technical discussions have continued even after your defense, let's agree to keep it up. Also, thank you for the LaTeX template that this thesis is based on. Erik, please shut up, I'm already convinced that you are smarter than me, no need to rub it in (I'm joking of course, don't shut up). I'm also eternally thankful for your timely and thorough review comments. Martin, you are such a good lunchmate, workmate, and generally, an amazing person. I'm very happy that you decided to join us in the industry, at Advenica and in my team, no less. I'm not going to mention by name all the rest of the department, but if you know me then I know you and I'm happy to have worked alongside you, I really am. You are all great people, no exceptions.

I would like to round off with a thank you to my entire extended family for their support, encouragement and not to mention, their many questions. Mom and dad, thank you for no less than everything, you've defined me in all the things that matter. My good character traits are yours and my faults are my own. Liza, thank you for being a great sister, and also for the help with the cover image. And finally, I would like to thank Ellen, Ebba, Elliot, and Edvin for offering no particular help in any practical sense of the word but still, somehow, providing that decisive bit of motivating factor. Without you, this book and the results herein would lack meaning. Love, dad.

Alexander Nilsson
Lund, April 2023

Contribution Statement

The following papers are included in this dissertation:

Paper I A. Nilsson, T. Johansson, and P. Stankovski Wagner. "Error Amplification in Code-based Cryptography". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019.1 (Nov. 2018), pp. 238–258

Paper II J.-P. D'Anvers, Q. Guo, T. Johansson, A. Nilsson, F. Vercauteren, and I. Verbauwhede. "Decryption Failure Attacks on IND-CCA Secure Lattice-Based Schemes". In: *Public-Key Cryptography – PKC 2019*. Ed. by D. Lin and K. Sako. Cham: Springer International Publishing, 2019, pp. 565–598

This paper is the result of a merge of [DVV18] and [GJN19a].

Paper III Q. Guo, T. Johansson, and A. Nilsson. "A Key-Recovery Timing Attack on Post-quantum Primitives Using the Fujisaki-Okamoto Transformation and Its Application on FrodoKEM". in: *Advances in Cryptology – CRYPTO 2020*. Ed. by D. Micciancio and T. Ristenpart. Cham: Springer International Publishing, 2020, pp. 359–386

Paper IV A. Nilsson, I. E. Bocharova, B. D. Kudryashov, and T. Johansson. "A Weighted Bit Flipping Decoder for QC-MDPC-based Cryptosystems". In: *2021 IEEE International Symposium on Information Theory (ISIT)*. 2021, pp. 1266–1271

Paper V Q. Guo, C. Hlauschek, T. Johansson, N. Lahr, A. Nilsson, and R. L. Schröder. "Don't Reject This: Key-Recovery Timing Attacks Due to Rejection-Sampling in HQC and BIKE". in: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2022.3 (June 2022), pp. 223–263

This paper is the result of a pre-publication merge, with [HLS21], due to the independent discovery of the same weakness.

Paper VI Q. Guo, D. Nabokov, A. Nilsson, and T. Johansson. *SCA-LDPC: A Code-Based Framework for Key-Recovery Side-Channel Attacks on Post-Quantum Encryption Schemes*. Submission Pending. 2023

The table below summarizes the responsibilities Alexander had in each paper:

Paper	*Writing*	*Concepts*	*Implementation*	*Evaluation*
Paper I	●	◐	■	◐
Paper II	◐	○	●	○
Paper III	●	●	■	◐
Paper IV	●	●	■	●
Paper V	●	●	●	◐
Paper VI	●	◐	●	●

In the table, ■ indicates roles where Alexander took primary or sole responsibility for the given role while ● indicates the responsibility was shared with one or more co-authors. In contrast, ◐ indicates contribution without taking responsibility for the role.

In paper I, Alexander was given primary responsibility for implementation and responsibility for writing the simulation and result sections. The development of the concept was a group effort though the more senior co-authors took primary responsibility. The evaluation of the result was mostly handled by the co-authors.

In paper II, Alexander was introduced to the project after the concept was fully developed, but was given the task of helping with the implementation. The writing was confined to the sections relevant to the implementation. When the merge with [DVV18] was decided, Alexander did part of the writing.

In paper III, the basic concept was introduced by the senior co-authors and was more fully developed as a shared responsibility. The source-code vulnerability itself was discovered by Alexander who also took full responsibility for the implementation. The evaluation of the results was shared and the evaluation concerning prior work was primarily handled by the co-authors.

In paper IV, a similar concept to the final paper was developed by Alexander together with a co-worker (not co-author) but failed to deliver satisfactory results. The co-authors of the final paper provided crucial insight into making the concept work. Alexander took the primary responsibility for implementation. Writing and evaluation were evenly divided between the co-authors.

In paper V, Alexander found the vulnerability present in the two titular schemes, the concept was then further developed by the Lund co-authors into real and practical attacks. The concept and one of the two attacks were independently discovered and published on eprint [HLS21] by the non-LU-based co-authors. After contact was established between the two groups it was decided a merge would be beneficial to all parties. Alexander took full responsibility for implementing the attack against of one the two vulnerable schemes. Writing and evaluation were split evenly between all the co-authors.

In paper VI, the concept was introduced by the senior co-authors, Alexanders contribution to the concept was limited to his part of the paper. Alexander took primary responsibility of one half of the implementation. Writing and evaluation were shared between all co-authors.

More comprehensive descriptions of each paper's contributions are available in Section 7.1.

Other Contributions

The following works have also been published during Alexander's Ph.D. studies, but are not included in this dissertation. Listed by descending number of citations.

- A. Nilsson, P. N. Bideh, and J. Brorsson. *A Survey of Published Attacks on Intel SGX*. arXiv `https://arxiv.org/abs/2006.13598`. 2020
- Q. Guo, T. Johansson, and A. Nilsson. *A Generic Attack on Lattice-based Schemes using Decryption Errors with Application to ss-ntru-pke*. Cryptology ePrint Archive, Paper 2019/043. `https://eprint.iacr.org/2019/043`. 2019 (pre-merge version of paper II.)
- J. Brorsson, P. N. Bideh, A. Nilsson, and M. Hell. "On the Suitability of Using SGX for Secure Key Storage in the Cloud". In: *Trust, Privacy and Security in Digital Business*. Ed. by S. Gritzalis, E. R. Weippl, G. Kotsis, A. M. Tjoa, and I. Khalil. Cham: Springer International Publishing, 2020, pp. 32–47

The work done during this Ph.D. has been funded jointly by Advenica AB and by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

List of Abbreviations

This list gives the location of the first use of each abbreviation in this dissertation for Part I. Uses in Chapter 1 are ignored if they are repeated in Chapters 2 to 7.

AES	Advanced Encryption Standard	11
BF decoding	iterative Bit Flip decoding	32
BIKE	Bit flIpping Key Encapsulation	35
BLISS	Bimodal LattIce Signature Schemes	53
BP decoding	Belief Propagation decoding	32
BSC	Binary Symmetric Channel	29
CCA	Chosen Ciphertext Attacks	45
CRQC	Cryptographically Relevant Quantum Computer	24
CVP	Closest Vector Problem	39
DEM	Data Encapsulation Mechanism	17
DF oracle	Decryption Failure oracle	56
DFA	Decryption Failure Attacks	45
DFR	Decoding Failure Rate	36
DFT	Discrete Fourier Transform	44
DH	Diffie-Hellman	15
DHP	Diffie-Hellman Problem	16
DLP	discrete log problem	16
DSA	Digital Signature Algorithm	16
DSS	Digital Signature Scheme	16
ECC	Error Correcting Code	29
ECDH	Elliptic Curve Diffie-Hellman	17
ECDSA	Elliptic Curve Digital Signature Algorithm	17
FD oracle	Full-Decryption oracle	56
FO transform	Fujisaki-Okamoto transform	19
GCHQ	Government Communications HeadQuarters	14
HQC	Hamming Quasi-Cyclic	36
IND-CCA	INDistinguishability under adaptive Chosen Ciphertext Attack .	13

IND-CPA, INDistinguishability under Chosen Plaintext Attack . . . . , 12
ISD, Information Set Decoding . . . . , 32
KEM, Key Encapsulation Mechanism . . . . , 17
LDPC, Low-Density Parity-Check . . . . , 32
LWE, Learning With Errors . . . . , 39
MAC, Message Authentication Code . . . . , 11
MDPC, Moderate-Density Parity-Check . . . . , 32
MLWE, Module Learning With Errors . . . . , 39
MS decoding, Min-Sum decoding . . . . , 32
NIST, National Institute of Standards and Technology . . . . , 26
NONCE, Number used only ONCE . . . . , 12
NTT, Number Theoretic Transform . . . . , 44
PC oracle, Plaintext-Checking oracle . . . . , 49, 56
PKC, Public Key Cryptography . . . . , 14
PKE, Public Key Encryption . . . . , 15
PQC, Post-Quantum Cryptography . . . . , 26
pqRSA, post-quantum RSA . . . . , 24
QC-MDPC, Quasi-Cyclic Moderate-Density Parity-Check . . . . , 32
QKD, Quantum Key Distribution . . . . , 25
RLWE, Ring Learning With Errors . . . . , 39
RMRS, Reed-Müller and Reed-Solomon . . . . , 36
ROM, Random Oracle Model . . . . , 13
SCA, Side-Channel Attack . . . . , 51
SIKE, Supersingular Isogeny Key Encapsulation . . . . , 27
SP decoding, Sum-Product decoding . . . . , 32
SVP, Shortest Vector Problem . . . . , 39
TRNG, True Random Number Generator . . . . , 14

Contents

Overview of Research Field

Introduction

CRQCs, or Cryptographically Relevant Quantum Computers, have been theorized to threaten all modern ciphers, to one degree or another. In 1994 Peter Shor [Sho94] published his seminal paper "Algorithms for quantum computation: discrete logarithms and factoring". This work completely undermines the security of, as it is now known, classical Public Key Cryptography (PKC), by taking advantage of the properties of quantum mechanics. Later Grover devised the quantum search algorithm [Gro96] which, in theory [Gro97; Ber10], affects not only PKC but also symmetric ciphers, the second category of cipher schemes.

The lack of fully realized CRQCs has in the general sense rendered the threat non-immediate. It is even less of an issue for symmetric cryptography since it would be enough to simply double the key size since this would keep the quantum attack complexity on par with the previous classical security level.

For the area of PKC, the situation is a degree more dire and requires special attention from the cryptographic community. Quantum safe cryptography, more commonly referred to as Post-Quantum Cryptography (PQC), concerns itself with the design and cryptanalysis of new replacement PKC algorithms. PQC algorithms are designed to be secure against both classical attacks and quantum attacks.

The non-immediacy of the threat of CRQCs does not imply that there is no urgency to plan ahead, since it can take decades to complete the migration from classical PKC to next-generation PQC algorithms. There are many time-consuming steps to be taken, from the first design of a new encryption or digital signature scheme to evaluation, standardization, implementation, deployment, and finally deprecation of old systems. A simple, almost trivial, model to reason about the urgency of migration is known as "Mosca's Theorem" [Mos18]. It states, simply put, that if

$$x + y > z$$

then you have a problem. Here x is the number of years your data has to remain protected, y is the number of years necessary to migrate the relevant systems to quantum-safe cryptography. Finally, z is the number of years until CRQCs are fully realized.

In an effort to stay ahead of the looming threat, the American National Institute of Standards and Technology (NIST) launched in 2016 a standardization project for PQC algorithms [NIS16]. The idea was to invite the global academic cryptography community to submit proposals for next-generation quantum-safe public key encryption (PKE) and digital signature (DS) schemes. The intention, then, was to focus the community towards the analysis and evaluation of these algorithms such that at the end of the project a narrow selection would be announced. This transparent process would offer academia, industry, and governments alike a high degree of confidence in the claimed security of each scheme.

At the time of this writing the project has whittled down the original 63 proposals to 4 (1 PKE + 3 DS schemes) accepted for standardization. A final selection from the remaining 3 PKE schemes will be done in the fourth and final round [Ala+22].

The research area of PQC is relatively new and the ultimate fear is the discovery of a cryptanalytical breakthrough that allows either classical computers or CRQCs to break whole families of the underlying computational hardness assumptions, upon which the security of the schemes depend. This would be devastating to any affected scheme. It is desirable therefore that any future cryptographic standard offers alternatives in regard to different families of underlying hardness assumptions in the hope that if one fails, we will already have standardized alternatives that are still secure.

Citing a lack of diversity, NIST has announced a new call for proposals [NIS22] for DS schemes in the hope that a new scheme might fill some gaps for certain use cases and offer alternatives in terms of what forms the mathematical foundation of its security.

There are a few different families, or categories, of encryption schemes [Ala+19]. As already stated, these are divided based on what computational hardness assumption they rely upon. The two most common hardness assumption families are Lattice-based cryptography and Code-based cryptography. Within both, there are some different variations to each.

While both families rely on different mathematical principles most family members share a few properties relevant to this dissertation, these being ciphertext malleability [Sho01] and decryption failures [Sch+22b; Ara+22; Alb+22; Agu+22]. The former property lets us know that the ciphertext can be modified in a way so that the effects on the plaintext can be predicted. Sometimes this takes the form of multiple ciphertext encodings that still decrypts to an identical plaintext. The latter property refers to the fact that these schemes suffer from a non-zero probability of decryption failures. Both of these properties can have negative effects on a PKE's security. For instance, it has been shown that decryption failures leak some bits of information about the secret-key [Bol+14; HGS99; HS00; JJ00; How+03; GN07; Flu16; Din+16; GJS16; Ber+18].

A Decryption Failure Attack (DFA) is an attack where the attacker exploits the malleability of a scheme and has access to a Decryption Failure oracle (DF oracle) that, somehow, lets the attacker know whether or not a decryption failure occurs for that particular ciphertext (which can be arbitrarily crafted). DFAs are classified under the Chosen Ciphertext Attack (CCA) attack model, though the assumption of a DF oracle is not quite as strong as the full decryption oracle available in the CCA model [Sma16].

For schemes that only offer security against the lesser Chosen Plaintext Attack (CPA) model, it follows naturally that they would be vulnerable against CCA type of attacks, such as DFAs. It is common then, for CPA secure public-key encryption schemes to apply a generic transform [HHK17; FO99] to create a CCA secure encryption scheme. Indeed, this is how the encryption schemes described in this dissertation have been constructed since it simplifies the security proofs greatly [Sch+22b; Ara+22; Alb+22; Agu+22; Che+20].

These attack models are very important tools for cryptanalysis [Sma16], but they do not capture the intricacies of real-world software or hardware implementations. Even if an encryption scheme, under some clearly defined assumptions, is mathematically proven to be CCA secure it may still be vulnerable to attacks. In fact, both hardware and software realizations of any encryption scheme are almost certainly vulnerable somehow, if naively implemented [KS05].

All implementations, unless specifically protected, leak information about the inner state of the algorithm. If the leakage has any dependency on a secret (key or message), it can be used to mount a so-called side-channel attack (SCA) [Koc96]. The side-channel can take the form of timing [BB03; Str10; BT11; Str13; Bru+16; Kau+16; DAn+19; Waf+19; PT19], power [KJJ99; Ngo+21; Ham+21; GJJ22; Sch+22a] or electromagnetic emanation [Rav+20; GLG22] variations. Some theoretical SCA attacks are generic in that they do not depend on the characteristics of the leakage medium [Uen+22]. There are even examples of side-channels based on acoustic measurements from the high-frequency "coil whine" generated by capacitors inside the CPU [GST14; GST17]. The most powerful SCAs based on timing variations require no physical access to make leakage measurements and can therefore be mounted remotely.

How PQC PKE schemes, even though nominally CCA secure, might be attacked by decryption failure attacks using side-channels, is the main topic of this dissertation.

1.1 Dissertation Outline

This dissertation is organized as follows. Part I contains the "Kappa", Chapters 1 to 7. Part II contains the included publications, referenced as papers I to VI. Finally Part III contains the Popular Science Summary in Swedish.

The "Kappa" is first introduced in Chapter 1, where we are now. Here are also some notations and typesetting conventions introduced. Classical cryptography is summarized in Chapter 2 where upon Chapter 3 follows with information on the effects of quantum computing and some general introduction to PQC. Chapters 4 and 5 continues with some more details on coding- and lattice-based cryptographic schemes, respectively. Chapter 6 finally gets close to the topic of this dissertation and presents the relevant related works. Chapter 7 is the final chapter and summarizes the entire Ph.D. project and the included publications.

General Dissertation Focus. Chapters 2 to 6 will give a discussion of the background information necessary to comprehend this dissertation in a meaningful way. The idea is that these chapters will make the included publications easier to follow, compared to reading the individual papers independently. This means that mathematical strictness is intentionally sacrificed in favor of readability, and in-depth discussions are sometimes sacrificed in favor of putting the work in a larger context. It is my hope that this initial intent survives contact with the reality that is your experience of the printed/electronic book in front of you.

1.2 Notations and Typesetting Conventions

In this dissertation, new acronyms are collated into the list on page xii, with page references. Only the first appearances in Chapters 2 to 7 are referenced. Here follows some mathematical concepts and notations necessary to follow along in this dissertation.

Let N be a positive integer and let a be any integer. We denote the modulo operation as the postfix operator

$$a = r \mod N, \tag{1.1}$$

so that r is the least non-negative remainder such that $a - r$ is a multiple of N. We define the set $\mathbb{Z}_N$ as the set of all integers modulo N. For any set S we use $\#(S)$ to denote the number of elements in the set. We say that two integers are relatively prime, or coprime, if their only common positive integer factor is 1. If a and N are coprime there exists an integer a^{-1} such that

$$aa^{-1} = 1 \mod N, \tag{1.2}$$

then we say that a^{-1} is the multiplicative inverse of a in $\mathbb{Z}_N$.

We note that the triple $(\mathbb{Z}_N, \cdot, +)$ is a commutative finite ring since it satisfies closure, associativity, existence of an identity element, invertibility, and commutativity for both of the two operations addition and multiplication. If p is a prime, a finite field of characteristics p is denoted $\mathbb{F}_p$ and we have that $\mathbb{F}_p = \mathbb{Z}_p$.

A set of polynomials in X whose coefficients are elements of $\mathbb{F}_p$ is denoted $\mathbb{F}_p[X]$ which also forms a ring with the natural definitions of addition and multiplications of polynomials, with modulo p for the coefficients. Polynomials, f, are typeset in the non-italics font face to distinguish them from scalar values, s. A finite ring can be defined as $\mathbb{F}_p[X]/\mathrm{f}(X)$ where all polynomials are given modulo $\mathrm{f}(X)$. For example, if $\mathrm{f}(X) = X^N + 1$, we write $\mathbb{F}_p[X]/X^N + 1$, then $N - 1$ is the maximum polynomial degree.

Consider the selection $p = 2$ and $\mathrm{f}(X) = X^4 + 1$. In this case one could write the following example, (note the coefficients are either one or zero, due to reduction modulo p):

$$\begin{aligned} \mathrm{h}(X) \cdot \mathrm{g}(X) &= (1 + X + X^2)(X + X^3) \mod X^4 + 1 \\ &= X + X^3 + X^2 + X^4 + X^3 + X^5 \mod X^4 + 1 \\ &= 1 + X^2. \end{aligned} \tag{1.3}$$

Just as with integers modulo N, when N is prime one obtains a finite field, so is $\mathbb{F}_p[X]/\mathrm{f}(X)$ also a finite field, if f is an irreducible polynomial. Being a finite field, we know there always exits an inverse g^{-1} to any polynomial $\mathrm{g} \in \mathbb{F}_p[X]/\mathrm{f}(X)$. In cases where it is clear that the operations are taking place in a stated polynomial finite field, the mod notation is omitted.

Often, some randomness is needed and the sampling operation of a value x, from a set of values $\mathbb{X}$, is given by the following operator:

$$x \xleftarrow{\$} \mathbb{X}, \tag{1.4}$$

the shape of the distribution is uniform randomness unless otherwise stated in the surrounding text.

Matrixes and vectors are typeset in bold face, $\mathbf{M}, \mathbf{v}$. Matrices are always written with capital letters. The identity matrix is denoted $\mathbf{I}^{n \times n}$ where $n \times n$ denotes the number of rows and columns of the matrix. For example,

$$\mathbf{I}^{4\times 4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{1.5}$$

The transpose of a matrix $\mathbf{A}$ is denoted $\mathbf{A}^{\mathrm{T}}$. Two vectors can be concatenated and this is illustrated by $\mathbf{a}||\mathbf{b}$.

The support of an n-bit binary vector $\mathbf{v}$ in $\mathbb{F}_2^n$ is the set of indexes of all non-zero positions v_i of the vector:

$$\mathit{supp}(\mathbf{v}) = \{i : v_i \neq 0,\ 0 \leq i \leq n\}. \tag{1.6}$$

The Hamming weight is the number of non-zero positions in a binary vector:

$$w_H(\mathbf{v}) = \#(\mathit{supp}(\mathbf{v})). \tag{1.7}$$

The Hamming distance is the number of positions that differs in two binary vectors $\mathbf{v}$ and $\mathbf{w}$:

$$d_H(\mathbf{v}, \mathbf{w}) = w_H(\mathbf{v} + \mathbf{w}). \tag{1.8}$$

For the sake of brevity and readability, functions and algorithms are defined with $:=$ and listed using the following non-traditional form:

$$\mathsf{Foo}(a, b, c) := \left\{ \begin{array}{|l} \text{Statement1} \\ \text{Statement2} \\ \text{Statement3} \\ \downarrow \end{array} \right. . \tag{1.9}$$

Where statements 1, 2 and 3 are each executed, in order, for alg. Foo, with inputs a, b and c.

Classical Cryptography

2.1 Cryptography and Cryptanalysis

Cryptography is an important building block to the very foundation of our modern, digital, way of life. It started as the art of hiding meaning in writings, of making things confidential and to make communication eavesdropping proof. It is the art of using ciphers, to encrypt whatever it is that you have to say and letting only the intended recipient know your intent.

Today, cryptography is so much more. The subject has grown to encompass not only confidentiality but also integrity protection, authenticity, signature verification, interactive proofs, and secure multi-party computation, to highlight just a few.

It starts with an adversary. This adversary can be anyone. It can be hardened criminals, military combatants, cloud service providers, spies, the combined resources of governments, or simply your nosy script-kiddie next door. We, as cryptography practitioners, very seldom discuss who the adversary is, because they should not have the power to decipher your communications, no matter what. Let us name our adversary A and assume she is very powerful indeed.

Cryptanalysis refers to the art of analysis with the goal of discovering or disproving weaknesses in cryptographic systems. For each new attack technique discovered, it empowers our adversary A such that it is never possible to put the genie back in the bottle. It is a fact of life, therefore, that we must continuously adapt to an ever-increasing adversarial power.

2.1.1 Historical ciphers

Through the ages A has shown her many insidious faces in the most cunning ways. But more important to the topic of this dissertation is the arms race A's ever-increasing attack power is prompting us, the defenders, into. One of the earliest known examples of confidentiality protection is the development of the Atbash cipher [RV14], which was originally used to render Hebrew unintelligible except for those in the know. The key is simple and can have only one value; each letter in the alphabet was replaced by the letter at the same position in the alphabet, after reversing the sorting order, e.g. ab would have been replaced by zy, if the English/Latin alphabet was used.

Another early technique was the Scytale [Kel98] used by the early Greeks, the Spartans in particular. The Scytale was a simple piece of wood whose diameter provided a sort-of key as a piece of parchment or leather was twined around it, and without the correct Scytale, whatever was written would be disjointed and unreadable.

Then we have the well-known Caesar cipher [Sin03] which cannot be excluded from any list of historical ciphers. This cipher is quite the improvement over the Atbash cipher in that there are actually n possible keys, where n is the size of the alphabet. It works simply by taking each letter in the plain text (before encryption) and shifting it by x positions to the right in the alphabet. If the letter is shifted beyond the end of the alphabet we wrap around and continue counting from the letter *a*.

2.2 Modern Symmetric Constructions

Thus far we have seen ciphers that provide confidentiality protection of a message. The encryption and decryption process share the same key, which must be known to all participating parties. We call these symmetric ciphers.

Modern symmetric ciphers are split into block ciphers and stream ciphers. A stream cipher uses a key *k* by which it can encrypt the given plaintext. The output of the encryption is called the ciphertext. And the decryption uses ciphertext as input and produces the original plaintext as output. The encryption function must be reversible, otherwise, we would not get back the original plaintext. A block cipher uses inputs and outputs of size *b*, we call this the block size. A block cipher is usually used in the context of a surrounding construction known as the mode of operation, which enables the cipher to encrypt arbitrarily long messages. Some modes of operation turn a block cipher into a stream cipher.

There is also the possibility of integrity protection using a concept of a Hash function (H). These functions require no key and can accept arbitrary long inputs. The output of a hash function is of fixed length n and a cryptographically strong hash function must uphold the following properties:

Pre-image resistance Given a hashed value h, it should be hard to find a message m such that $\mathsf{H}(m) = h$.

Second pre-image resistance Given a message m_1, it should be hard to find another message m_2 such that $\mathsf{H}(m_1) = \mathsf{H}(m_2)$.

Collission resistance It should be hard to find any message pair m_1, m_2 such that $\mathsf{H}(m_1) = \mathsf{H}(m_2)$.

One step further is the concept of authentication, that is, verifying that the received data can only have been produced by someone who possesses knowledge of the authentication key. We call such a construction a Message Authentication Code (MAC). These are usually constructed from hash functions, but with an additional secret input, the authentication key.

2.3 Security Notions

From the earliest historical examples above it is quite easy to see that A can force her way through the ciphers simply by attempting each possible key. We call this technique brute-force and it can in theory be used to break any cipher which does not hold the property of perfect secrecy. This property, a concept due to Claude Shannon [Sha49], colloquially known as the father of information theory, is equivalent to information theoretic security; The ciphertext may hold no additional information about the plaintext (beyond its length) for A to uncover. This is possible only under the following conditions:

- The key is truly random.
- The key is of equal length as the plaintext.
- The key is used only once.

Intuitively, any valid plaintext can be constructed from a ciphertext, they are all equally likely given a truly random key. Such schemes are in fact unbreakable [Sha49], even in the face of a computationally unbounded[1] adversary A_∞. For more practical schemes with small keys, another measure of security is required.

Brute force is a way by which we can measure computational security, or computationally bounded security. For instance, a cipher is considered secure if the security level, λ, offers a large enough key search space 2^λ and there are *no known* attacks offering a significant advantage over the brute force technique. One of the most common ciphers in use today is the Advanced Encryption Standard (AES), of which the $\mathsf{AES\text{-}128}$ variant uses a key size of 128 bits. It is assumed that this cipher provides $\lambda = 128$-bits of security because there are no publicly known attacks that can do much better than a brute-force attack. To attempt a brute-force attack would require a decryption attempt with each of the 2^{128} possible keys (unless A gets lucky and finds an early match).

Semantic security is an analogy to the perfect secrecy notion under a computationally bounded adversary A_λ. Informally, it means that an adversary limited to 2^λ operations, cannot glean any additional information from the ciphertext about the plaintext, beyond its length. It has been shown by Goldwasser and Micali [GM19] that semantic security is implied by the INDistinguishability under

[1] I.e. unlimited brute-force ability

Chosen Plaintext Attack (IND-CPA) security model. IND-CPA is better suited for formal security proofs than the concept of semantic security [Bel+97].

Consider a cipher encryption function $E_k(m)$ where k, m is a key and a message, respectively. Ciphertext indistinguishability then refers to the inability of the adversary A_λ to distinguish between two encrypted messages of her own choosing. It is usually set up as a thought experiment, or a game, like the following:

1. Adversary A_λ receives access to an encryption oracle, $\mathcal{O}_{e_k}$, from the challenger. The encryption oracle is a construct that while encrypting any message provided to it and returning the resulting ciphertext, it will never reveal the encryption key k to the adversary.

2. Adversary A_λ may call $\mathcal{O}_{e_k}$ or perform any other computational operations within the computational bound, i.e. less than 2^λ operations.

3. Adversary A_λ presents any two distinct chosen plaintexts m_0, m_1 to the challenger.

4. The challenger selects a bit $b \xleftarrow{\$} \{0, 1\}$ uniformly at random. Next the ciphertext $c = E_k(m_b)$ is sent back to A_λ.

5. Adversary A_λ may perform any computational operations within the computational bound, i.e. less than 2^λ operations.

6. Adversary A_λ submits a guess b' of the value b.

7. Adversary A_λ wins the game if $b' = b$.

The security game as given above is more specifically known as the left-or-right indistinguishability notion [Bel+97], as the goal of A_λ is to determine which of the two plaintexts (left or right) was encrypted. We define the advantage of A_λ against the encryption scheme Π as

$$\mathrm{Adv}_{\Pi}^{\text{IND-CPA}} = 2 \cdot \left| \Pr\left[\mathsf{A}_\lambda \text{ wins}\right] - \frac{1}{2} \right|, \tag{2.1}$$

where Pr denotes the probability of a given [*condition*], a value between 0 and 1, inclusive. An encryption scheme Π is IND-CPA secure if $\mathrm{Adv}_{\Pi}^{\text{IND-CPA}}$ is small, for any computationally bounded adversary A_λ. That is, A_λ may receive at most only a negligible advantage over random guessing.

The reader might here point out that IND-CPA implies non-deterministic encryption. In other words, if the encryption is not random in some way, A_λ may simply request $c^* \leftarrow \mathcal{O}_{e_k}(m_0)$ in step 2. If $c = c^*$ in step 5, then in step 6 submit $b = 0$, or $b = 1$ otherwise. Non-deterministic encryption may be achieved by the inclusion of a Number used only ONCE (NONCE). In such cases we would redefine the encryption function as $E_k(m; r)$, where r is a NONCE.

Ciphertext malleability of an encryption scheme is a property commonly held by IND-CPA secure schemes. Consider a known ciphertext c corresponding to the encryption of the unknown plaintext m. If it is possible for an adversary A_λ to find a ciphertext c^*, such that the relation of m and m^* is known and computationally feasible for the attacker to compute, the cipher is said to be malleable.

Put differently, it is easy to see in the IND-CPA security game that if the relation $m = f(m^*)$ is within the computational bound (i.e. less than 2^λ operations) then a malleability attacker would break the security of the scheme *if* the attacker could somehow gain knowledge of the triple (f, c^*, m^*). The relation f may be publicly known for the encryption scheme, so that leaves the decryption of c^* into m^*. However, the ability for A_λ to decrypt c^* is only possible with access to a decryption oracle. This is not present in the IND-CPA model. For many use cases, IND-CPA is quite sufficient due to the setting in which the encryption scheme is deployed.

However, to properly capture the power of a malleability attack, or other[2] attacks that go beyond the chosen plaintext attacks, it is necessary to expand the attack model. This is done by giving A_λ access to a decryption oracle $\mathcal{O}_{d_k}$ in steps 1, 2 and 5 in addition to the already present encryption oracle $\mathcal{O}_{e_k}$. INDistinguishability under adaptive[3] Chosen Ciphertext Attack (IND-CCA) is such a model. In this attack model, it is necessary to limit $\mathcal{O}_{d_k}$ such that it will offer no decryption of the specific challenge ciphertext c (step 4), otherwise it would trivially allow A_λ to win the game.

We have already seen how a malleability attacker makes use of the decryption oracle to attack malleable IND-CPA secure schemes. So, similarly to how IND-CPA security implied non-deterministic encryption, IND-CCA security implies non-malleable encryption. Being the stronger security model, it follows that any IND-CCA secure scheme is always IND-CPA secure, too.

2.3.1 Regarding Hash Functions

The computational security of a hash function is ideally given in terms of its output size n. For example, finding a pre-image by brute force is done within 2^n calls to the hash function. And by the brithday paradox, a collision is very probable after hashing $2^{n/2}$ messages. Due to the risk of collisions, the output size of the hash function is usually given as $n \geq 2\lambda$ for λ-bit security level.

When encryption schemes are mathematically proven secure in the IND-CPA or IND-CCA models, they are usually also done so under the Random Oracle Model (ROM). The ROM is an ideal mathematical construction, used instead of a concrete hash function. This simplifies the security proofs greatly, and it functions like the following:

[2] For examples, see Chapter 6.

[3] There is an intermediary *non-adaptive* attack model that we skip over in this dissertation.

- There is a black box. The box has unlimited memory and access to a True Random Number Generator (TRNG).
- The box takes an arbitrarily long sequence of bits m as input.
- If m is not previously seen before, the box uses the TRNG to produce a truly uniform random number h. Both h and m is remembered.
- If the box has already seen the input m before, it uses its memory to return the same output h it returned last time.

So the ROM functions like an ideal hash function, the problem is that we do not know how to construct an ideal hash function. The real hash functions we use in practice are only approximations of a random oracle. This means, if a cipher proven IND-CCA secure under ROM is actually insecure, then the security problem exists in the hash function[4], and not in the cipher itself. I.e., many security proofs are shown to be as good as the security of the selected hash function.

2.4 Public Key Cryptography

To communicate confidentially over an open channel with someone whom you have never previously exchanged any shared secret with was an ability assumed to be impossible, for about 2000 years or so [Sin03]. With only symmetric cryptography available, both communicating parties must each have a copy of the secret key, which is used both to encrypt and decrypt.

"The Possibility of Non Secret Digital Encryption" was worked out by James H. Ellis [Ell70] at the British Government Communications HeadQuarters (GC-HQ). He figured that there must be a way, following the principles he laid out, for encryption and decryption to require different keys. In this manner, the encryption key (coupled with a specific user) may be published in a directory for anyone to read. Thus encryption can be performed by anyone. Decryption, on the other hand, can only be done by the user holding the corresponding secret decryption key. Ellis had invented the field of Public Key Cryptography (PKC) and the principles of asymmetric ciphers. Soon thereafter, Clifford Cocks, a coworker of Ellis, discovered a way to construct a practical scheme following those same principles. Malcolm J. Williamson, another coworker of Ellis, followed suit and developed the principles behind the world's first key exchange algorithm. Unfortunately, all of this was kept secret from the public eye for 27 years, before the British government declassified the works in 1997 [Sin03].

Fortunately, Whitfield Diffie and Martin Hellman realized the same possibilities in their seminal paper *"New directions in cryptography"* and invented the key

[4] or in whatever other assumptions made by the proof, see for example the side-channel discussion given later in Section 6.2.

exchange scheme known as Diffie-Hellman (DH) [DH76]. This discovery, even though more than 5 years later, is independent of the works of Ellis, Cocks, and Williamson. Ron Rivest, Adi Shamir, and Leonard Adleman invented the RSA algorithm [RSA78] shortly thereafter, the very first practical Public Key Encryption (PKE) scheme.

The main insights of Ellis, (and Diffie & Hellman) were the following: Suppose f is a one-way function such that it is easy to compute and hard to compute its inverse f^{-1}. Moreover, suppose f^{-1} *can* be easily computed by knowledge of some secret information. In such cases, f is called a trapdoor function and the secret information is the key to the trapdoor. Assuming the existence of such functions, Ellis conjectured the possibility of encryption by $c = f_{pk}(m)$ and decryption $m = f_{sk}^{-1}(c)$, where *pk* contains the public parameters used for encryption and *sk* contains the trapdoor information used for decryption. The understanding is that f^{-1} is infeasible to compute without the knowledge of the secret key *sk* (trapdoor information) [Ell70; DH76].

The insight of Clifford Cocks (and Rivest, Shamir & Adleman) was finding such a trapdoor function:

$$c \leftarrow E_{pk}(m) = m^e \pmod N, \tag{2.2}$$

where public key $pk = (e, N)$ and N is a large composite number $N = pq$, with p and q both primes. Here, e is the public exponent used for encryption. It must be relatively prime to the Euler totient function of N, which is $\phi(N) = (p-1)(q-1)$ since both p and q are prime. In other words, the greatest common divisor (gcd) of the public exponent and $\phi(N)$ must be one:

$$\text{gcd}(e, (p-1)(q-1)) = 1. \tag{2.3}$$

A common [Sma16] selection for e is a small prime like 3, 17, or 65537 since the constraint then holds for all choices of p, q.

The inverse of the encryption function is the decryption function:

$$m \leftarrow \underbrace{E_{pk}^{-1}(c)}_{\text{Hard to compute}} = \underbrace{D_{sk}(c) = c^d \pmod N}_{\text{Easy to compute knowing } d}, \tag{2.4}$$

where $sk = (d, p, q)$. Here, d is the private decryption exponent derived during the key generation process. It should satisfy [Sma16]:

$$e \cdot d = 1 \pmod{(p-1)(q-1)}. \tag{2.5}$$

As it stands, we have now informally presented the main intuitions of the "naive RSA" encryption scheme [Sma16]. This non-complete scheme is both deterministic and malleable, thus preventing it from being IND-CPA and IND-

CCA secure. To achieve those properties more work is needed, but it serves to illustrate the main point well enough: Inverting the encryption function, Eq. (2.2), was supposed by Williamson and Rivest, Shamir & Adleman, a difficult problem, and despite many years of concentrated research the supposition still holds. Here, the trapdoor information is d, and by extension the prime factorization of N [Sma16].

The RSA PROBLEM states that knowing only c, e and N, find m such that Eqs. (2.2) and (2.3) hold. The FACTOR PROBLEM states that given only a composite N of two primes, factor into primes p and q. A solution for the latter can be used to solve the former, thus we say that there is a reduction from the RSA PROBLEM to the FACTOR PROBLEM. In other words, if an algorithm exists that can solve the FACTOR PROBLEM, then the RSA PROBLEM can be solved [Sma16].

It is not known if the opposite reduction exists, that is, if one can solve the RSA PROBLEM, can one then factor a composite number? Related though, it can be shown that *computing the secret exponent* d can be reduced to the FACTOR PROBLEM and vice versa; the FACTOR PROBLEM can be reduced to the computation of d. This means that the two problems are complexity theoretically equivalent [Sma16].

To summarize, a secret-key recovery attack[5] is just as hard as the FACTOR PROBLEM. A message recovery attack[6] may *possibly* be easier to compute, due to the RSA PROBLEM. No such publicly known algorithms for solving the RSA PROBLEM exist though [Sma16].

2.4.1 Other Asymmetric Constructions

The RSA algorithm can also be used as a Digital Signature Scheme (DSS). It works by swapping the public and private keys so that the private key is used to sign (i.e. encrypt) a message. Then anyone can use the public key to verify the signature (i.e. decrypt and check for equality). Digital signatures are the analogue to MACs and are used for message or sender authentication, but it does not require the sharing of a symmetric key.

In parallel with the RSA PROBLEM, another common mathematical problem is the DISCRETE LOG PROBLEM (DLP); In any group G and $b \in G$, powers b^k, is defined for all integers k. The integer k then is known as the discrete logarithm $\log_b(a) = k$ such that $b^k = a$. The DLP, the computation of such a k, seems to be a difficult problem. Though there are several groups where this is easy, in the general case, we know of no efficient algorithm. The DH key exchange algorithm relies on the DIFFIE-HELLMAN PROBLEM (DHP) which can be reduced to DLP. The DLP also serves as a foundation for the operation and security of the Digital Signature Algorithm (DSA), a DSS. Normally, DH uses a multiplicative group of integers, modulo a large prime p.

[5] Recovery of a secret key parameter (d, for RSA).

[6] Recovery of a single value of m.

Elliptic curves function as alternative groups for the DLP and offer shorter keys for the same security level. Elliptic Curve Diffie-Hellman (ECDH) serves as an alternative to regular DH. Elliptic Curve Digital Signature Algorithm (ECDSA) serves as an alternative DSS to either RSA or DSA.

2.5 KEMs and DEMs

While PKC solves many problems related to key distribution and key scheduling, it is not a good solution for encryption of bulk data. Such schemes are simply too inefficient. Instead, a symmetric cipher is used for data encryption, whereas some asymmetric cipher is used to protect the symmetric encryption key. This setup is what we call the KEM/DEM approach, which is also known as a hybrid[7] scheme. A KEM is a Key Encapsulation Mechanism and refers to the selected PKE scheme. A DEM is a Data Encapsulation Mechanism and refers to the symmetric encryption scheme, it can be your favorite block or stream cipher. Three algorithms make up a KEM, these are

- $(pk, sk) \leftarrow \mathsf{KeyGen}()$ — Generates public and secret key-pairs, at random.
- $(c, k) \leftarrow \mathsf{Encaps}_{pk}()$ — Using public key pk, a uniformly random symmetric key $k \in \mathbb{K}$ is generated and then encapsulated (encrypted) into the ciphertext c. The key space $\mathbb{K}$ is the set of all possible symmetric keys.
- $k \leftarrow \mathsf{Decaps}_{sk}(c)$ — Using secret key sk and the previously created ciphertext c, the key k is decapsulated (decrypted).

The concept then, is for the encryptor to acquire the public key of the recipient, and use the Encaps_{pk} function to generate a ciphertext c and a symmetric key k. The key is used to either encrypt a one-time payload (object encryption) or to set up a secure communication channel based on symmetric encryption primitives. For both options, the ciphertext c must be first transmitted before decryption will be possible.

The decryption procedure is to decapsulate the ciphertext using the Decaps_{sk} function and the secret key sk. Now the object or the channel can be decrypted with the symmetric key k.

As an aside, digital signatures have a similar same kind of construction where one signs not the actual message, but only the hash of the message, since that is much shorter and of a fixed length. Also, hash algorithms are extremely efficient, in comparison to any asymmetric algorithm.

[7] Not to be confused with a hybrid classical and post-quantum scheme.

The security game for the KEM construction must be slightly modified when dealing with KEMs, as opposed to regular PKEs. This is because there is no plaintext input/output anymore and the output of the encapsulation is instead a symmetric DEM key and a ciphertext. Also, considering that we are now exclusively discussing asymmetric schemes, there is no longer any need for the encryption oracle, since the adversary can encrypt by herself using the public key. The schema for the security game for IND-CCA security of KEMs is given below without further ado.

1. The challenger generates (or has previously generated) secret and public keys

   $$(pk, sk) \leftarrow \mathsf{KeyGen}()$$

   and two symmetric keys k_0, k_1. The first symmetric key is generated uniformly at random from the symmetric key space

   $$k_0 \overset{\$}{\leftarrow} \mathbb{K}$$

   and the second symmetric key is given by the KEM encapsulation function

   $$k_1, c^* \leftarrow \mathsf{Encaps}_{pk}()$$

2. The challenger picks a bit $b \overset{\$}{\leftarrow} \{0, 1\}$ uniformly at random, in secret, and then transmit pk, k_b and c^* to the adversary A_λ.

3. Adversary A_λ may construct any ciphertext c and call a decapsulation oracle $\mathcal{O}_{d_{sk}}$ for access to the corresponding DEM key

   $$k \leftarrow \mathcal{O}_{d_{sk}}(c) = \mathsf{Decaps}_{sk}(c)$$

   or perform any other operation, within the computational bound, i.e. less than 2^λ operations. The oracle $\mathcal{O}_{d_{sk}}$ is of course prohibited from decapsulating the given challenge ciphertext c^*, otherwise the game can be won trivially.

4. Adversary A_λ submits a guess b' of the value b.

5. Adversary A_λ wins the game if $b' = b$.

The adversary's advantage in the above game is defined in the same way as for IND-CPA security of PKE:

$$\mathrm{Adv}_{\Pi}^{\text{IND-CCA}} = 2 \cdot \left| \Pr\left[\mathsf{A}_\lambda \text{ wins}\right] - \frac{1}{2} \right| . \tag{2.6}$$

Thus the KEM scheme Π is secure if the advantage $\mathrm{Adv}_{\Pi}^{\text{IND-CCA}}$ is small. Put equivalently, the KEM is secure if the adversary can only gain a negligible advantage over random guessing when attempting to distinguish between a purely random key and a DEM key produced by Π.

The previous discussions on ciphertext malleability remain relevant; IND-CCA security still implies non-malleability. Note however, the encapsulation and computation of the challenge ciphertext c^* is honest as it is performed by the challenger and the key produced in concert is always random, guaranteed by Π. Therefore, we no longer need to have the notion of non-deterministic encryption for IND-CPA (KEM) security, due to how the KEM and the security game are constructed.

2.6 FO Transform

Proving IND-CCA security of a PKE scheme is often an involved and complicated proposition. Much easier is proving IND-CPA security and then applying a generic transform on top. One such transform was first proposed in 1999 by Eiichiro Fujisaki and Tatsuaki Okamoto in [FO99]. This Fujisaki-Okamoto transform (FO transform) was later revised by the same authors in [FO13]. The concept of KEM/DEM hybrid schemes was integrated with the FO transform by Dent [Den03], which spawned multiple equivalent variations and further security reduction improvements, by Hofheinz, Hövelmanns and Kiltz in [HHK17]. That work in particular served as a basis for many IND-CCA security claims of some newer generations of PKE schemes. Most recently, Hövelmanns, Hülsing, and Majenz [HHM22] made public a paper showing, among other things, updated security bounds of the variants of the FO transform with explicit rejections. Being the most natural variant of the transformation [HHM22], we start by describing it in the remainder of the current section.

Using any weakly secure non-deterministic PKE scheme Π as the basis of the transformation we have the triple $\Pi := \left(\mathsf{KeyGen}, \mathsf{Encrypt}_{pk}, \mathsf{Decrypt}_{sk}\right)$. The first step of the transform is to construct a deterministic but non-malleable scheme $\Pi^{\mathsf{G}} := (\mathsf{KeyGen}, \mathsf{Encrypt}_{pk}^{\mathsf{G}}, \mathsf{Decrypt}_{sk}^{\mathsf{G}})$ such that

$$\mathsf{Encrypt}_{pk}^{\mathsf{G}}(m) := \mathsf{Encrypt}_{pk}(m; \mathsf{G}(m)), \tag{2.7}$$

where $\mathsf{G}(m)$ is a ROM hash function that replaces the NONCE that was introduced for IND-CPA security, back in Section 2.3, page 12. $\mathsf{G}(m)$ fulfills the concept of random coins which ensures that the encryption function is deterministic, yet returns unrelated and unpredictable ciphertexts whenever m changes.

We define $\mathsf{Decrypt}^{\mathsf{G}}_{sk}$ in the following,

$$\mathsf{Decrypt}^{\mathsf{G}}_{sk}(c) := \left\{ \begin{array}{|l} m' \leftarrow \mathsf{Decrypt}_{sk}(c) \\ c' \leftarrow \mathsf{Encrypt}^{\mathsf{G}}_{pk}(m') \\ \textbf{if } m' = \perp \textbf{ or } c \neq c' \textbf{ output } \perp \\ \textbf{else output } m' \end{array} \right. , \tag{2.8}$$

where $\perp$ indicates an explicit failure message. Note in the above, the call to $\mathsf{Encrypt}_{pk}$ ensures the non-malleability of the scheme by comparing the received ciphertext with its re-encryption. This result is intuitive for the two cases of ciphertext modifications $c^* \leftarrow f(c)$. The first case is where the resulting decryption leads to a possibly related but still distinct value of $m' \neq m$. Consider the properties of the hash function G; Any change to its input, no matter how small, completely changes its output, in a way that cannot be predicted (under the random oracle model). Thus, the condition $c \neq c'$, must hold with overwhelming probability. The second case is where f ensures $m' = m$ and it is also caught by the comparison, though in this case the difference of the ciphertexts $c \neq c'$ may be arbitrarily small. These two cases are discussed more in-depth in paper III.

From the deterministic and non-malleable scheme Π^{G} it is now possible to complete the Fujisaka Okamoto transform to construct $\mathsf{FO}^{\perp}[\mathsf{G},\mathsf{H}] := (\mathsf{KeyGen}, \mathsf{Encaps}_{pk}, \mathsf{Decaps}^{\perp}_{sk})$, a KEM construct of

$$\mathsf{Encaps}_{pk}(\cdot) := \left\{ \begin{array}{|l} m \xleftarrow{\$} \mathbb{M} \\ c \leftarrow \mathsf{Encrypt}^{\mathsf{G}}_{pk}(m) \\ k \leftarrow \mathsf{H}(m) \\ \textbf{output } (k, c) \end{array} \right. \tag{2.9}$$

and

$$\mathsf{Decaps}^{\perp}_{sk}(c) := \left\{ \begin{array}{|l} m' \leftarrow \mathsf{Decrypt}^{\mathsf{G}}_{sk}(c) \\ \textbf{if } m' = \perp \textbf{ output } \perp \\ \textbf{else output } k \leftarrow \mathsf{H}(m') \end{array} \right. . \tag{2.10}$$

H and G represent two domain separated ROM hash functions. Domain separation means that they with overwhelming probability produce distinct values, even if given identical input.

It is also possible to define a similar transform $\mathsf{FO}^{\not\perp}[\mathsf{G},\mathsf{H}]$ with implicit rejection. This construct is very similar except it does not return any explicit error value

$\perp$. Instead, $\mathsf{Decaps}_{sk}^{\not\perp}$ with implicit rejection is defined as follows:

$$\mathsf{Decaps}_{sk}^{\not\perp}(c) := \left\{ \begin{array}{|l} m' \leftarrow \mathsf{Decrypt}_{sk}^{\mathsf{G}}(c) \\ \textbf{if } m' = \perp \textbf{ output } k \leftarrow \mathsf{H}(\sigma) \\ \textbf{else output } k \leftarrow \mathsf{H}(m') \end{array} \right. \downarrow \quad , \tag{2.11}$$

where σ is a random value stored in, or derived from, the secret key *sk*. In this transformation, the designer hides decapsulation failures by creating an unrelated symmetric key that will not match the Encaps_{pk} output value.

The Quantum Age

3.1 Quantum Computation

Consider a system of x possible subatomic particle positions. That gives us 2^x possible configurations where each position is either filled by a particle or not. Due to quantum level interference where each subatomic particle may affect all other particles in a wave-like manner, such systems are very hard to model on a classical computer in a way that does not scale exponentially with the problem size. In fact, if one would want to understand the quantum evolution of such a system it cannot be done without tracking every possible configuration, since it is not feasible to use statistical sampling methods, due to the interference.

In nature, large and complex quantum systems keep on evolving according to the laws of quantum mechanics, whether we can model or engineer facsimiles of them or not. So then, might one flip the problem on its head and instead utilize the subatomic particle/waveform duality and its interactions to solve complex computational problems, faster than conventional computers? (It is a leading question, the answer is yes.)

The fundamental piece of information, due to Shannon [Sha49], is a single bit. A qubit in a quantum system can, similarly, be described as being either 0 or 1, but it can also be in a superposition of these two values. A qubit thus represents the most fundamental information-carrying object in a quantum system. When measured, a qubit's state ”collapses” to either 0 or 1, but the outcome depends on the probabilities of each state. Quantum entanglement is the concept where if one measures one of two entangled qubits, one automatically gains some knowledge of the state of the other qubit, i.e. the state probabilities are connected and collapse at the same time.

To compute with a quantum system of multiple logical[1] qubits one may first initialize the system into a uniform superposition of all possible solutions to the

[1] Logical qubits differ from physical qubits in that that they are stable over time and do not suffer from effects such as quantum decoherence. A logical qubit may be constructed from multiple physical qubits with the aid of quantum error correction algorithms.

input problem. Then a series of quantum gates form a so-called quantum circuit, doing operations on all superpositioned configurations simultaneously. After completing the quantum circuit the state probabilities are no longer uniform and, if correctly implemented, the correct solution is likely to be given when measured.

Quantum computation offers time complexity reduction to some problems but it does not provide advantages over conventional computation in the general sense. Also, problems which are undecidable in classical computing are also undecidable with quantum computation.

3.2 Quantum Apocalypse

In 1994, Peter Shor published his seminal paper ”Algorithms for quantum computation: discrete logarithms and factoring” [Sho94]. In it, Shor capitalizes on the quantum phenomenons described above, to solve the integer factorization and discrete logarithm problems such that it scales polynomially with the problem size.

Time complexity concepts such as ”polynomial time” alluded to above, have not been discussed so far in this thesis. But it is sufficient to know that exponential time algorithms scale, well, exponentially, in runtime with the size of its input. On the other hand, polynomial time scales much slower since the scaling function is expressed as a polynomial, albeit with an arbitrarily high degree. Sub-exponential time is an intermediary classification that is not-quite exponential, but still more expensive to solve than polynomial time problems, asymptotically.

In other words, quantum computers have the ability to run Shor’s algorithm and thereby solve the integer factorization problem, not in sub-exponential time (like the best classical methods), but in polynomial time. Shor’s algorithm works by evaluating a periodic function on a superposition of all inputs within a wide range. Then applying a quantum Fourier transform, one obtains an approximate superposition of periods of the function. Measuring this superposition results in a random period, but the correct answer is more probable. Thus by applying the algorithm several times, the correct answer can be determined. One of the key insights of Shor’s was to use the modular exponentiation function, the period of which informs on the factors of the composite number. With the period, the factors can then be computed using a classical algorithm.

A quantum computer with enough logical qubits to solve the RSA problem we call a Cryptographically Relevant Quantum Computer (CRQC). Postulating that CRQCs will exist in the near future, then [Ber+17b] suggests that the RSA public parameters should scale to the size of *one gigabyte* in size, to remain secure[2]. Though, in this case the public modulus n is a composite of not two, but 2^{23} unique 1024-bit primes. This modified RSA scheme is called post-quantum RSA (pqRSA). Doing it this way implies 2^{110} quantum gates using $\approx 2^{34}$ qubits to

[2] The submission also mention one terabyte keys as a “feasible” option, presumably depending on one’s security requirements.

break pqRSA, which should be safe even in the long term. Needless to say, there are some usability concerns with pqRSA which would mandate users to look elsewhere for their encryption needs. DH, ECDH, or ECDSA offer no safe havens either since Shor's algorithm works just as well on discrete logarithms over both finite multiplicative groups of integers and elliptic curve groups.

Grover's algorithm [Gro96] is another threat posed by quantum computers. This algorithm can be more generally applied to break any encryption scheme, though the impact is more limited. Grover's algorithm can be most easily explained as a way of speeding up a search through an unsorted database to find one specific entry. Sometimes it is better explained as finding the roots of any generic function f using $\sqrt{N}$ instead of N evaluations of f. It works by initializing the quantum state into a uniform superposition of all possible input values (the database, or inputs to the function). The details of the algorithm itself remain outside the scope of this thesis, but the gist of it is two operations. One operation marks the correct answer in the database, and one amplifies this answer in the quantum state, making it more likely to be the final output once the quantum state collapses. After repeating these two operations $\sqrt{N}$ times, the correct answer will stand out from the rest of the probabilities.

The consequence of the quadratic speedup provided by Grover's algorithm is that the number of bits of security provided by symmetric encryption schemes is, in effect, halved. That is, AES with a 128-bit key can in theory be broken by 2^{64} quantum operations. If true[3], the obvious solution then is to use 256-bit keys for symmetric ciphers.

While the consequences appear manageable for symmetric encryption, just about all asymmetric cryptography that has been in use up to today will be severely affected by the coming of CRQCs (due to Shor's algorithm) and thus replacements must be sought. The rest of this chapter will discuss some techniques to this effect.

3.3 Quantum Cryptography

As mentioned, pqRSA does not appear to be a viable choice due to the size of the public key as well as the slow execution time of key generation, encryption, and decryption. However, quantum technologies can not only be used to break cryptography but can also be used as a physics-based foundation of security. The key idea is that the encoding of information into quantum states can be done in a way that is impossible to copy and intercept without detection.

Quantum Key Distribution (QKD) capitalizes on this idea in order to securely generate a secret key, to be used by quantum-safe symmetric ciphers, by sending a series of quantum particles, such as photons, between two parties. Upon receiving these particles they are measured to determine their quantum state, which cannot

[3] There is evidence [NIS18] to suggest that it might not be. Some say that Grover's algorithm will not achieve the full quadratic speedup, in practice, due to its serial nature, among other things.

be done by a third party without detection by the original parties. That is, on a physics level, one can be sure of the absence of eavesdroppers, and thus the security of the generated symmetric key.

In short, QKD can be used to establish new symmetric keys that can then be used by classical symmetric algorithms. The main drawbacks are the requirements of a pre-existing authenticated (but not encrypted) communication channel between the two parties as well as the obvious dedicated physical link (e.g. fiber cable) between the same.

3.4 Post-Quantum Cryptography

Considering that neither pqRSA nor QKD appears to be practical paths forward in the face of potential CRQCs one must look elsewhere for replacement algorithms of RSA, DH, DSA and elliptic curve-based variants.

Quantum safe cryptographic algorithms, better known as Post-Quantum Cryptography (PQC), is the term used that encompasses the next generation of asymmetric encryption solutions. PQC algorithms are encryption algorithms running on classical computers, designed to be impervious to quantum computers.

Right now a number of standardization efforts are underway, with the purpose to unify and focus the research community toward the goal of achieving trustworthy and performant PQC algorithms. Premier among those is the PQC standardization project by the American National Institute of Standards and Technology (NIST).

The research area PQC and the standardization project PQC are, in the general and specific case, working towards the next generation of asymmetric cryptographic standards, specifically KEM and DSS algorithms. The PQC project is a transparent endeavor but the ultimate power resides with NIST. In the first round, 2016, 63 proposals were accepted as fulfilling the submission requirements. This prompted the global cryptography research community to focus much of their attention on breaking or otherwise evaluating these proposals. The number of surviving proposals has been reduced after each of the first three rounds. As of this writing, there are now one key encapsulation mechanism and three digital signature schemes selected for standardization. Additionally, there is a fourth and final round to select one or two additional KEM algorithms from the remaining three surviving candidates.

The KEM scheme to be standardized is:

- CRYSTALS-Kyber

The remaining candidates for possible standardization are:

- Classic McEliece

- BIKE

- HQC
- ~~SIKE~~ – Withdrawn due to [CD23].

The following strong KEM schemes are no longer candidates for the PQC project, but are relevant to this dissertation:

- FrodoKEM
- NTRU

This dissertation does not consider signature schemes, therefore they are not listed, although they are indeed a major part of the upcoming standard.

As shown in Section 2.4, the security of an asymmetric encryption algorithm must be based on a trapdoor function of good renown. That is, easy to calculate one way, and expensive to invert without knowledge of the trapdoor key. For instance McEliece, BIKE, and HQC, listed above, are based on error-correcting codes which will be discussed in more detail in Chapter 4. The remaining three mentioned schemes are of course Kyber, FrodoKEM, and NTRU which use trapdoor functions related to lattices, the topic of Chapter 5.

Earlier in the NIST PQC project, a few more categories of trapdoor functions could be found among the candidates. These include hash-based, multivariate-based and schemes based on supersingular elliptic curve isogenies, but they are not very relevant to this dissertation. Though, the latter category includes Supersingular Isogeny Key Encapsulation (SIKE), which should be noted for being selected as a Round 4 finalist until it was withdrawn due to a new attack [CD23]. SIKE operated on supersingular elliptic curves, which are a special kind of elliptic curves that are not smooth and continuous, but rather curves which define only a limited number of points. Isogenies are special mappings between different ellipitic curves, which preserve some important group properties. The security of SIKE was dependent on the hardness of finding such isogenies.

The risks of an unexpected cryptanalytical breakthrough that affects an entire family of algorithms, such as lattice-based schemes, must be mitigated. In fact, this is the ultimate reason why NIST elected to keep only non-lattice schemes for possible standardization, side-by-side with Kyber, in round four. The idea is that if such a breakthrough does occur, there will already be a standardized replacement algorithm ready.

Cryptography based on Coding Theory

This chapter provides background knowledge of ciphers based on coding theory.

4.1 Introduction to Coding Theory

By adding redundancy to messages via special encodings one can be certain to successfully decode a received codeword and recover the transmitted message, even in the presence of noisy channels. These channels can have many different characteristics, but the most important case for this dissertation is the Binary Symmetric Channel (BSC). The BSC model is a memory-less channel whose transmitted bits are all equally likely to flip, according to some probability p.

Coding theory is used in many places, such as CDs, WiFi/radio, wired communication, error-correcting memory in servers, hard drives, and satellite communications. Practically in all places where storage or transmission must remain robust in the face of read or transmission errors. In quantum computation, error correction plays a central role due to the inherent noisiness of quantum mechanics.

There are many ways to construct such Error Correcting Codes (ECCs) and they may be specially tailored for the error rate and other characteristics of the channel model. The core principle remains the same however, by adding redundancy one lowers the transmission rate R of the channel but increases the number of errors that can be reliably handled.

Usually, one may encrypt a k-symbol message with r-symbols of redundancy resulting in a $n = k + r$ symbol long codeword. Such an encoding has the rate $R = k/n$ and this means that per single symbol of information that is to be transmitted, the number of transmitted symbols is actually $1/R$.

Taking a k-symbol message $\mathbf{m} \in \mathbb{F}_q^k$ one may construct the corresponding valid codeword by multiplying with a generator matrix $\mathbf{G} \in \mathbb{F}_q^{k \times n}$:

$$\mathbf{c} = \mathbf{m}\mathbf{G}. \tag{4.1}$$

The shape and structure of the generator matrix are specific to the encoding scheme in question, but in general, we say that it is of systematic form if the first $k \times k$ part is comprised of an identity matrix. That is if $\mathbf{G} = [\mathbf{I}||\mathbf{Q}]$. A systematic generator matrix has the property that the transmitted message is placed verbatim at the start of the codeword such as

$$\mathbf{c} = \begin{bmatrix} m_0 & m_1 & \dots & m_{k-1} & c_k & c_{k+1} & \cdots & c_{k+r-1} \end{bmatrix}, \tag{4.2}$$

where c_i for $k \leq i < k + r$ is the added redundancy symbols.

In this chapter we consider only messages with binary symbols ($q = 2$) and the (n, k)-codes $\mathcal{C}$ we have begun to describe can also be described as a k-dimensional subspace of $\mathbb{F}_2^n$. In other words, the space of k dimensions is described, redundantly, in n dimensions such that the only valid codewords are given by $\mathcal{C}$.

We say that the minimum distance of all valid codewords

$$d = min(\{d_H(\mathbf{a}, \mathbf{b}) : \forall\, \mathbf{a}, \mathbf{b} \in \mathcal{C}, \mathbf{a} \neq \mathbf{b}\}) \tag{4.3}$$

gives the error detection limit $d - 1$ of the code $\mathcal{C}$. The error correction limit is often denoted $t = (d - 1)/2$.

Next, we introduce the parity check matrix $\mathbf{H} \in \mathbb{F}_2^{r \times n}$ such that it represents the nullspace or kernel of the code $\mathcal{C}$. That is,

$$\mathbf{c}\mathbf{H}^{\mathrm{T}} = \mathbf{0} \tag{4.4}$$

must be fulfilled for all valid codewords. For instance, consider the parity check $\mathbf{H} \in \mathbb{F}_2^{4 \times 8}$ corresponding to a (8, 4) block code

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}. \tag{4.5}$$

The following equations must be satisfied for a valid codeword

$$\begin{array}{ccccccccc} & v_1 & & + & v_3 + & v_4 & & & + & v_7 = 0 \\ v_0 + & v_1 + & v_2 & & & + & v_5 & & & = 0 \\ & & v_2 & & & + & v_5 + & v_6 + & v_7 = 0 \\ v_0 & & & + & v_3 + & v_4 & + & v_6 & & = 0 \end{array}. \tag{4.6}$$

All codes related to this discussion are linear, which means that the sum of two codewords

$$\mathbf{c}_0 + \mathbf{c}_1 = \mathbf{m}_0\mathbf{G} + \mathbf{m}_1\mathbf{G} = (\mathbf{m} + \mathbf{m})\mathbf{G} \tag{4.7}$$

is also a valid codeword. Same with the parity check matrix:

$$\mathbf{H}(\mathbf{c}_0 + \mathbf{c}_1) = \mathbf{H}\mathbf{c}_0 + \mathbf{H}\mathbf{c}_2 = \mathbf{0} + \mathbf{0} = \mathbf{0}. \tag{4.8}$$

According to the channel model (often the BSC), an error $\mathbf{e}$ is introduced, during transmission, to the codeword such that the received vector $\mathbf{v} = \mathbf{c}+\mathbf{e}$ differs from what was originally transmitted. The syndrome $\mathbf{s}$ of the received vector $\mathbf{v}$ is given by

$$\mathbf{s} = \mathbf{v}\mathbf{H}^{\mathrm{T}} = (\mathbf{c} + \mathbf{e})\mathbf{H}^{\mathrm{T}} = \mathbf{e}\mathbf{H}^{\mathrm{T}}. \tag{4.9}$$

That is, a valid code word, with no symbol errors, results in the all-zero syndrome given in Eq. (4.4). Note how the result of each equation in Eq. (4.6) corresponds to the value at the same position s_i of the syndrome. That is, each non-zero position of the syndrome is the result of an unsatisfied parity check.

A Tanner graph is a certain kind of bi-partite graph, useful for visualization of linear codes. It displays the relationship between code symbols and the parity-check sums that check on them. The example $\mathbf{H}$ given in Eq. (4.5) have the corresponding Tanner graph representation in Fig. 4.1. Here we see that the symbols are given by elements v_i of the received vector. In this example, each symbol is connected to two parity check equations.

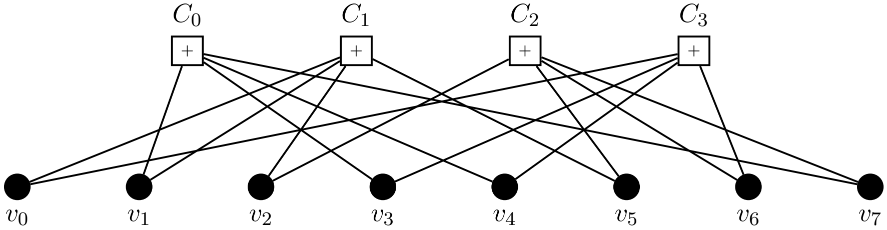


Figure 4.1: The Tanner graph representation of $\mathbf{H}$

The Tanner graph representation informs us on the relationship of each given symbol in the received vector (called a variable node, at the bottom of the graph) and we may use these to check the validity of each symbol through its parity check equation (called a check node, at the top of the graph).

Tangential to this discussion, is the concept of source compression coding (or simply source coding). This is a way of utilizing coding theory where instead of adding redundancy to a message one instead transmits *only* the redundancy symbols. Thus, a low entropy message can be reconstructed from fewer symbols and thereby achieving a compression scheme. Source coding is utilized in paper VI.

The DECODING PROBLEM, serves as the security foundation of code-based schemes. It requires the adversary to find the closest codeword $\mathbf{c} \in \mathcal{C}$ to a given received vector $\mathbf{v} \in \mathbb{F}_2^n$. Let $\mathbf{v} = \mathbf{c} + \mathbf{e}$ and assume there exists a closest codeword. Note that finding $\mathbf{e}$ is equivalent to finding $\mathbf{c}$.

The SYNDROME DECODING PROBLEM is to compute a minimal weight $\mathbf{e}$ given $\mathbf{s}$ such that $\mathbf{He} = \mathbf{s}$. Note that the syndrome decoding is equivalent to the regular DECODING PROBLEM since a received vector can be constructed easily by assuming the all-zero codeword and a systematic form generator matrix. This is because a parity check matrix can easily be constructed from a systematic form generator matrix:

$$\begin{aligned} \mathbf{G} &= \begin{bmatrix} \mathbf{I} & \mathbf{Q} \end{bmatrix} \\ \mathbf{H} &= \begin{bmatrix} \mathbf{Q}^{\mathrm{T}} & \mathbf{I} \end{bmatrix} \end{aligned}. \tag{4.10}$$

Here the trailing identity matrix in $\mathbf{H}$ corresponds to the r-bit redundancy where each bit corresponds to a single parity check equation. Thus

$$\mathbf{v} = \begin{bmatrix} 0 & 0 & \dots & 0 & s_0 & s_1 & \dots & s_{r-1} \end{bmatrix} \tag{4.11}$$

is a valid received vector for the given syndrome.

Considering only large matrices $\mathbf{H}$, general decoding for random codes is a very hard problem. Information Set Decoding (ISD) is one general algorithm, but it runs in exponential time. Though, by careful design of the code $\mathcal{C}$ and the parity check matrix $\mathbf{H}$ in particular, decoding can be performed efficiently. Code-based cryptography exploits this difference in order to construct a one-way trapdoor function.

Relevant to this dissertation is primarily Low-Density Parity-Check (LDPC) codes and variants Moderate-Density Parity-Check (MDPC) and Quasi-Cyclic Moderate-Density Parity-Check (QC-MDPC) codes. We will briefly mention binary Goppa codes also, in relation to the first described code-based cryptosystem, McEliece in Section 4.2.

4.1.1 LDPC codes and their decoding

LDPC codes due to Gallager [Gal62] are linear block codes that are constructed using sparse Tanner graphs and they have very good decoding properties. This construction is their primary distinguishing factor, namely the sparseness of the Tanner graph. Even for huge codes, the number of connected edges, to each node, is very small. Consequently, the number of non-zeroes per row of $\mathbf{H}$ is also small.

The example LDPC code given in Fig. 4.1 is not very good, for three reasons. First, it is very small and will therefore not achieve very good error correction performance. Second, it is not a sparse graph, mainly due to its size, so it is not really an LDPC. And third, it contains short cycles of length 4. For example, you may revisit node v_3 by $v_3 \rightarrow C_0 \rightarrow v_7 \rightarrow C_2 \rightarrow v_3$. A good performing LDPC code on the other hand will be long, of low row weight and lack short cycles.

These codes can be decoded using various decoding techniques such as Belief Propagation decoding (BP decoding), Sum-Product decoding (SP decoding), Min-Sum decoding (MS decoding) approximation and iterative Bit Flip decoding (BF decoding). We will here focus on BF decoding also due to Gallager [Gal62].

1. Decoding begins by calculating the syndrome **s** as per Eq. (4.9). If it is equal to the zero vector, stop the decoding.

2. For each connected variable node (symbol in the received vector), count the number of failed parity checks f_i (syndrome bits).

3. Identify the set $\mathcal{S}$ of symbols where f_i is large.

4. Flip the symbols (bits) in the set $\mathcal{S}$.

5. Repeat steps 1 to 4 until we are done (all parity checks are satisfied) or we have reached a pre-determined maximum number of iterations.

If the pre-determined maximum number of iterations in step 5 is reached, we declare a decoding failure. How the set $\mathcal{S}$ in step 3 is determined is subject to a number of different strategies, but the most common in regular BF decoding is to use an empirically determined threshold δ like so $\mathcal{S} = \{f_i : f_i \geq \delta, 0 \leq i < n\}$.

4.2 McEliece

In 1978 Robert J. McEliece proposed [McE78] the very first PKE based on a hard problem from coding theory. This system is built upon random binary Goppa codes and the ciphertext is comprised of a valid codeword plus a random error vector.

A seemingly random generator matrix is the public key (details will follow) and by selecting a random codeword and error vector the adversary finds it difficult to recover the error-free codeword. The decoding can proceed efficiently only by knowledge of the underlying structured code in the secret key.

The security level of the McEliece system has remained stable despite many efforts to attack it for the last 40+ years. The system has also prompted much other related work, most notably by Neiderreiter [Nie86] who proposed a more efficient variant while preserving the security of the scheme.

Let Γ be a choice of $\mathcal{C}$, a length n random binary Goppa code [McE78] of code dimension k with minimum distance $d = 2t + 1$. Here $t \approx r/\log_2(n)$ which gives us the additional public parameters $n = 1024$, $k = 524$, $r = 500$ and $t = 50$ from the original paper [McE78].

The secret key is comprised of Γ with the corresponding structured generator matrix $\mathbf{G}$ and the two additional matrices $\mathbf{P} \in \mathbb{F}_2^{n\times n}$ and $\mathbf{S} \in \mathbb{F}_2^{k\times k}$. $\mathbf{P}$ is a permutation matrix and $\mathbf{S}$ must be invertible such that $\mathbf{S}\mathbf{S}^{-1} = \mathbf{I}$. The parameters n, k, r, t are public. We also require an efficient decoding algorithm for Γ. The public key then is $\mathbf{G}' = \mathbf{SGP}$, which is a $k \times n$ generator matrix for a new, unstructured looking, code.

The message $\mathbf{m}$ is encrypted like so:

$$\mathbf{c} = \mathbf{m}\mathbf{G}' + \mathbf{e}, \tag{4.12}$$

where **e** is a random error vector of Hamming weight $w_H(\mathbf{e}) = t$.

The decryption proceeds as follows. Compute

$$\mathbf{cP}^{-1} = \mathbf{mG'P}^{-1} + \mathbf{eP}^{-1} = (\mathbf{mS})\mathbf{G} + \mathbf{eP}^{-1}. \tag{4.13}$$

Since **P** is strictly a permutation matrix, it is now possible to use the efficient decoder for the structured code Γ to remove the noise $\mathbf{eP}^{-1}$ and recover **mS**. Then it is a simple matter of calculating the plaintext by $\mathbf{m} = \mathbf{mSS}^{-1}$

Lacking the secret key, the adversary's task is to decode **c** to the nearest codeword $\mathbf{mG'}$ which is the hard general DECODING PROBLEM, if $\mathbf{G'}$ does not expose any structure.

The Neiderreiter variant used by the Classic McEliece submission to the NIST PQC standardization effort will now be described, very briefly, just to highlight the differences.

The scheme uses the invertible matrix $\mathbf{S} \in \mathbb{F}_2^{r\times r}$ and the permutation matrix $\mathbf{P} \in \mathbb{F}_2^{n\times n}$, similar to before. This variant uses a parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{r\times n}$ for the sampled binary Goppa code Γ. The secret key is **H**, **P** and **S**. The public key is the scrambled parity-check matrix $\mathbf{K} = \mathbf{SHP} \in \mathbb{F}_2^{r\times n}$.

Encryption does away with the Generator matrix by assuming the all-zero code word. Instead, the plaintext is simply a random error vector **e** of weight $w_H(\mathbf{e}) = t$. The ciphertext then is the syndrome

$$\mathbf{s} = \mathbf{Ke}. \tag{4.14}$$

Here the adversary is required to solve the t-error correcting problem for the scrambled parity check matrix. This is equivalent to the DECODING PROBLEM given by the original McEliece variant.

Decryption using the secret key is done by computing

$$\mathbf{S}^{-1}\mathbf{s} = \mathbf{S}^{-1}\mathbf{Ke} = \mathbf{S}^{-1}(\mathbf{SHP})\mathbf{e} = \mathbf{H}(\mathbf{Pe}). \tag{4.15}$$

Plainly, it is now possible to use a syndrome decoder for **H** to find **Pe**, since $w_H(\mathbf{Pe}) = t$, and thus the plaintext $\mathbf{e} = \mathbf{P}^{-1}\mathbf{Pe}$.

A CCA secure KEM scheme can be had by using the general FO transform on either the original or the Neiderreiter variant.

The advantage of the McEliece scheme over other PQC algorithms is that it offers small ciphertexts, efficient encryption/decryption, and a very good (and long) security analysis track record. The main disadvantage however is the slow key generation and considerable storage and memory requirements of the public key.

4.3 BIKE

The Bit flIpping Key Encapsulation (BIKE) KEM scheme is a code-based cryptosystem based not on random Goppa codes, but rather structured quasi-cyclic moderate density parity check codes that were designed to improve the size disadvantage of the McEliece cryptosystem.

MDPC codes have a slightly higher $w \approx \sqrt{n}$ number of non-zeroes in the parity check matrix than LDPC codes, which affects decoding negatively, but have other properties useful for cryptography. In order to reduce the storage size requirement of the matrices the QC-MDPC variant uses binary cyclic submatrices of square dimensions in its parity check matrix. For example, a QC-MDPC code of rate $R = 1/n_0$ is given by

$$\mathbf{H} = \begin{bmatrix} \mathbf{H_0} & \dots & \mathbf{H}_{n_0} \end{bmatrix}, \tag{4.16}$$

where

$$\mathbf{H_j} = \begin{bmatrix} h_{j,0} & h_{j,1} & \dots & h_{j,k-1} \\ h_{j,k-1} & h_{j,0} & \dots & h_{j,k-2} \\ \vdots & & \ddots & \vdots \\ h_{j,1} & h_{j,2} & \dots & h_{j,0} \end{bmatrix} \quad j \in \{0, 1, \dots, n_0\}. \tag{4.17}$$

We see that $\mathbf{H_j}$ can be completely represented by the first row $\mathbf{h_j}$ of each submatrix, using only a rotating shift operation for the subsequent rows.

The BIKE cryptosystem is constructed as a Neiderreiter variant of McEliece, instantiated by QC-MDPC codes of $n_0 = 2$, $R = 1/2$, with each cyclic block of size r and with error correction limit $t \approx \sqrt{n}$. The private key is $\mathbf{H}$, represented by $(\mathbf{h_0}, \mathbf{h_1})$, each of length r and such that $w_H(h_0) = w_H(h_1) = w/2$.

Due to the QC-MDPC construction, there is no need for any scrambling or permutation matrices. The public key is

$$\mathbf{h} = \mathbf{h_1 h_0^{-1}}. \tag{4.18}$$

Of course, $\mathbf{h_0}$ must be invertible, which is ensured during key generation.

The plaintext is a random error vector $\mathbf{e} = (\mathbf{e_0}, \mathbf{e_1})$, of weight $w_H(\mathbf{e}) = t$ and the ciphertext is:

$$\mathbf{s} = \mathbf{e_0} + \mathbf{e_1 h}. \tag{4.19}$$

Decryption with the secret key is calculated as

$$\mathbf{sh_0} = \mathbf{e_0 h_0} + \mathbf{e_1 hh_0} = \mathbf{e_0 h_0} + \mathbf{e_1 h_1}, \tag{4.20}$$

which can be recovered back to the original $\mathbf{e} = (\mathbf{e_0}, \mathbf{e_1})$ with a syndrome decoder.

Decryption without the secret key requires the adversary to solve a variant of the general SYNDROME DECODING PROBLEM since the public key h appears unstructured.

The public keys are more than a hundred times smaller compared to McEliece's due to the structure of the QC-MDPC codes. However, BIKE occasionally suffers from decoding failures. That is, by some small probability the decoder fails to recover the error vector from a valid ciphertext. This is due to the probabilistic nature of the BF decoder for these kinds of LDPC-derived codes. The rate of failures is called the Decoding Failure Rate (DFR).

To construct an IND-CPA or IND-CCA secure KEM, some additional operations are necessary. The idea, if not the particulars, are the same as previously discussed for the $\mathsf{FO}^{\not\perp}[\mathsf{G},\mathsf{H}]$ transform in Section 2.6. Those details are described in the specification [Ara+22] and to some extent in paper V. It is relevant to note that the IND-CCA security is conditional on the DFR being lower than $2^{-\lambda}$. This has been shown by modeling and extrapolating empirical simulations to be the case, but there is as yet no proof [Ara+22], which is why only use cases where IND-CPA security is enough are recommended.

4.4 HQC

The Hamming Quasi-Cyclic (HQC) KEM scheme [Agu+22; Agu+18] takes a different approach than McEliece and BIKE. It depends on two types of codes, a public decodable $(n,\ k)$-code $\mathcal{C}$ used for the trapdoor and correct decryption purposes and a random $(n,\ 2n)$ quasi-cyclic (actually double circulant) code for security guarantees.

The code $\mathcal{C}$ must be publicly agreed upon, before starting the key generation. This is currently defined as a Reed-Müller and Reed-Solomon (RMRS) shortened and concatenated code. The details of this RMRS code will not be discussed here, as the scheme would work for any code with error-correcting capacity δ. Other public parameters are the set of polynomials $\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$, code size n and the target Hamming weights w, w_e and w_r.

The key generation is comprised of

- The quasi-cyclic, uniformly random, parity check matrix $\mathbf{h} \overset{\$}{\leftarrow} \mathcal{R}$.
- The two vectors $(\mathbf{x}, \mathbf{y}) \overset{\$}{\leftarrow} \mathcal{R}^2$ of low weight $w_H(x) = w_H(y) = w$
- The syndrome $\mathbf{s} = \mathbf{x} + \mathbf{hy}$.

The secret key is $(\mathbf{x}, \mathbf{y})$ and the public key is $(\mathbf{h}, \mathbf{s})$.

The encryption function first samples $\mathbf{e}$, $\mathbf{r}_1$ and $\mathbf{r}_2$ such that $w_H(e) = w_e$ and $w_H(r_1) = w_H(r_2) = w_r$. Encryption then is mainly comprised of the two equations

$$\mathbf{u} = \mathbf{r}_1 + \mathbf{hr}_2 \tag{4.21}$$

and

$$\mathbf{v} = \mathbf{mG} + \mathbf{sr}_2 + \mathbf{e}, \tag{4.22}$$

where Eq. (4.21) is the calculation of a noisy syndrome, as generated by $\mathbf{h}$. This constitutes a quasi-cyclic variant of a general (i.e. hard) SYNDROME DECODING PROBLEM. Equation (4.22) represent the plaintext encoded by the public code $\mathcal{C}$ which is rendered undecodable by moving past the decoding threshold δ of $\mathcal{C}$, due to $\mathbf{s}$. The ciphertext is the tuple $(\mathbf{u}, \mathbf{v})$.

Decryption is done by running the public decoder

$$\mathbf{m} = \mathcal{C}.\mathsf{Decode}(\mathbf{v} - \mathbf{u}\mathbf{y}), \tag{4.23}$$

which works because

$$\begin{aligned} w_H(\mathbf{v} - \mathbf{u}\mathbf{y}) &\leq \delta \\ w_H((\mathbf{x} + \mathbf{h}\mathbf{y})\mathbf{r}_2 - (\mathbf{r}_1 + \mathbf{h}\mathbf{r}_2)\mathbf{y} + \mathbf{e}) &\leq \delta \\ w_H(\mathbf{x}\mathbf{r}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}) &\leq \delta \end{aligned} \tag{4.24}$$

is true with overwhelming probability. The DFR is upper bounded by a theoretical analysis of the error vector distribution of $\mathbf{x}\mathbf{r}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}$.

In other words, an attacker without knowledge of the secret key is faced with either 1) decoding the code word $\mathbf{v}$, which is infeasible due to the noise being greater than δ, i.e. the closest code word is not the correct one. Or 2) decoding the syndrome $\mathbf{u}$, due to the random parity check matrix $\mathbf{h}$, which is also infeasible as previously discussed. Or 3) solve $\mathbf{s}$ for $\mathbf{x}$ or $\mathbf{y}$, which is also an infeasible SYNDROME DECODING PROBLEM.

HQC is, as is BIKE and McEliece, an IND-CPA secure PKE scheme, turned IND-CCA secure KEM by way of a variant of the FO transform. However, unlike BIKE, HQC is IND-CCA secure without provisional conditions, since the specification already includes a precise theoretical analysis of the DFR upper bound.

Lattice-based Cryptography

This chapter relates, in part, to the seminal work of Ajtai and Dwork [AD97] where the authors gave the first cryptographic constructions whose security properties followed from the hardness of various problems on lattices. This work kicked off a number of successive improvements and refinements to the proposed cryptographic systems. The most common and successful schemes, as judged from the perspective of NIST PQC standardization effort, are based on the LEARNING WITH ERRORS (LWE) problem, as originally defined by Regev [Reg05].

NTRU-based schemes are unrelated to the Ajtai-Dwork lattice schemes and they differ from LWE on several points in both construction and pedigree. The common denominator between the NTRU- and LWE-based schemes is the fact that the security reductions trace back to the problem that relates to finding the closest or shortest vectors in an integer lattice of large dimensions. This dissertation does not pretend to offer any authoritative exposition on the subject of CLOSEST VECTOR PROBLEM (CVP) or SHORTEST VECTOR PROBLEM (SVP), for such information any textbook on the subject would suffice, see for example [Sma16], for a beginner-friendly introduction.

LWE schemes offer several important variations and can be classified into LWE, RLWE or MLWE schemes. In short:

LWE operates over vectors and matrices of integers modulo q, that is $\mathbb{Z}_q$.

RLWE or RING LEARNING WITH ERRORS operates over polynomials modulo f and modulo q for coefficients, that is $\mathbb{F}_q[X]/\mathrm{f}(X)$.

MLWE or MODULE LEARNING WITH ERRORS operates over vectors and matrices of polynomials modulo f and modulo q for coefficients.

In this chapter, we describe schemes based on NTRU, LWE (FrodoKEM) and MLWE (Kyber) problems.

5.1 NTRU

Here follows a brief description of NTRU[1], as it was described in the original 1996 paper [HPS96] due Hoffstein, Pipher and Silverman. Numerous improvements and changes have been proposed over the years since then. Most notably the NIST PQC standardization effort included three KEM-like NTRU variants; NTRU-Encrypt [Zha+17], NTRU-HRSS-KEM [Sch+17] and NTRU Prime [Ber+17a]. NTRUEncrypt and NTRU-HRSS-KEM later merged into the NIST-PQC submission NTRU [Zha+19]. The simplified version presented below is also denoted NTRU-HPS, which refers to the original authors, when it must be distinguished from other variants. A comparison of the NTRU-based NIST-PQC submissions can be found in [Sch18], due to Schanck. In NTRU all operations are being defined over the finite ring $\mathbb{F}_q[X]/X^N - 1$.

NTRU(-HPS) is parameterized over the coprime positive integers (N, q, p), N is the degree limit of the set of polynomials, q is a relatively large modulus often selected as a multiple of two and p is a small modulus, most often a prime. One common choice found in the literature is $(251, 128, 3)$. During key generation two polynomials f and g are randomly selected.

Polynomial f is saved in the secret key, and must additionally satisfy the requirement of being invertible under modulus q and modulus p. That is, one must find the inverses $\mathrm{f_p}$ and $\mathrm{f_q}$, like so:

$$\begin{aligned} \mathrm{ff_p} &= 1 \mod p \\ \mathrm{ff_q} &= 1 \mod q \end{aligned}. \tag{5.1}$$

The public polynomial h is given by

$$\mathrm{h} = p\mathrm{f_q g}. \tag{5.2}$$

Before a message can be encrypted, it must first be converted into a polynomial m of degree at most $N-1$, we limit the coefficients to mod p. This can be done in a number of different ways, but arguably the simplest is to reinterpret the binary encoding of the message such that each bit-value corresponds to the coefficient of their respective position in the message vector. I.e. a common binary encoding of "NTRU" (`01001110 01010100 01010010 01010101`) would be reinterpreted as

$$\begin{aligned} \mathrm{m} = \quad & x^{30} + x^{27} + x^{26} + x^{25} + x^{22} + x^{20} + x^{18} + \\ & x^{14} + x^{12} + x^{9} + x^{6} + x^{4} + x^{2} + 1 \end{aligned}. \tag{5.3}$$

This encoding has a simple conversion procedure, but it disregards the possibility of the $p-2$ other possible coefficients entirely, so it is not size efficient. The

[1] Some suggestions I have found are that the name is an abbreviation for one of "N-th degree Truncated polynomial Ring Units", "Number Theorists 'R' Us" or "Number Theory Research Unit". Or it is simply a given name.

NIST-PQC submissions NTRU and NTRU Prime both use more space-optimal encoding schemes. Regardless of the chosen encoding scheme, we now have our message polynomial m.

During encryption, a blinding polynomial r is randomly selected and multiplied with h in the public key. Finally, m is added to the result to produce the encrypted polynomial c. Thus,

$$\mathrm{c} = \mathrm{rh} + \mathrm{m} \tag{5.4}$$

is the ciphertext.

To decrypt, the polynomial a is calculated, such that

$$\mathrm{a} = \mathrm{fc} \mod q. \tag{5.5}$$

The polynomial b

$$\mathrm{b} = \mathrm{a} \mod p \tag{5.6}$$

is calculated. Finally, the decryption is done by

$$\mathrm{m} = \mathrm{f_p b} \mod p. \tag{5.7}$$

Recall that all operations are in the ring $\mathbb{F}_q[X]/X^N - 1$. Thus when calculating a the following equivalence holds

$$\begin{aligned} \mathrm{a} &= \mathrm{fc} \mod q \\ &= \mathrm{f(rh + m)} \mod q \\ &= \mathrm{f(r}p\mathrm{f_q g + m)} \mod q \\ &= \mathrm{ff_q}p\mathrm{rg + fm} \mod q \\ &= p\mathrm{rg + fm} \mod q \end{aligned} . \tag{5.8}$$

Note here that all coefficients are small in relation to q, thus no modulo reduction is actually needed. Assuming this to be the case,

$$\mathrm{a} = p\mathrm{rg} + \mathrm{fm} \tag{5.9}$$

becomes

$$\begin{aligned} \mathrm{b} &= \mathrm{a} \mod p \\ &= p\mathrm{rg + fm} \mod p \\ &= \mathrm{fm} \end{aligned} . \tag{5.10}$$

And final decryption is thus

$$\mathrm{f_p b} = \mathrm{f_p fm} = \mathrm{m} \mod p. \tag{5.11}$$

The security of the NTRU scheme and its variants has been the focus of much research. This dissertation will not repeat its state-of-the-art here. Though, the reader should be aware of a few important results, as follows.

To date, the most effective attack strategy against NTRU is to utilize so-called lattice-based attacks. This is why NTRU falls within the lattice-based category of PQC schemes.

More rigorous specifications than the one presented in this section, introduce a few more parameters and relations between them, thereby ensuring a few important properties. One of these properties is the (hopefully low, or non-existent) probability of large coefficients in the a polynomial. This probability directly affects the probability of decryption failures.

5.2 FrodoKEM

FrodoKEM is a conservative KEM scheme that focuses on security and simplicity. This cryptosystem relies on the (plain) LWE PROBLEM which relates to solving a noisy linear system modulo a known integer q. The LWE PROBLEM can be reduced to several known lattice problems [Reg05].

First, key generation starts from a secret seed from which matrices $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$, $\mathbf{S} \in \mathbb{Z}_q^{n \times \bar{n}}$ and $\mathbf{E} \in \mathbb{Z}_q^{n \times \bar{n}}$ are randomly generated, where n and $\bar{n}$ are public parameters of the scheme. The matrix $\mathbf{A}$ and

$$\mathbf{B} = \mathbf{AS} + \mathbf{E} \tag{5.12}$$

comprise the public key. $\mathbf{S}$ and $\mathbf{E}$ make up the private key.

To encrypt, matrices $\mathbf{S}'$, $\mathbf{E}'$ and $\mathbf{E}''$ must first be randomly sampled. The next step is to calculate

$$\mathbf{B}' = \mathbf{S}'\mathbf{A} + \mathbf{E}' \tag{5.13}$$

and

$$\mathbf{V} = \mathbf{S}'\mathbf{B} + \mathbf{E}''. \tag{5.14}$$

Then the message m is encoded with a suitable encoding $\mathsf{Frodo.Encode}(\cdot)$, explained below, and added to $\mathbf{V}$ so that the ciphertext c can be constructed, like so

$$c \leftarrow (\mathbf{C}_1, \mathbf{C}_2) = (\mathbf{B}', \mathbf{V} + \mathsf{Frodo.Encode}(m)). \tag{5.15}$$

To decrypt one simply calculates

$$\mathbf{M} = \mathbf{C}_2 - \mathbf{C}_1\mathbf{S} \tag{5.16}$$

and then $\mathbf{M}$ must be decoded back into the decrypted plaintext m' using the inverse encoding scheme:

$$m' = \mathsf{Frodo.Decode}(\mathbf{M}). \tag{5.17}$$

To check the correctness of decryption we can summarize the noise terms into $\mathbf{E}'''$ as follows

$$\begin{aligned} \mathbf{M} &= \mathbf{C}_2 - \mathbf{C}_1\mathbf{S} \\ &= (\mathbf{V} + \mathsf{Frodo.Encode}(m)) - (\mathbf{S}'\mathbf{A} + \mathbf{E}')\mathbf{S} \\ &= \mathsf{Frodo.Encode}(m) + \mathbf{S}'\mathbf{B} + \mathbf{E}'' - \mathbf{S}'\mathbf{A}\mathbf{S} - \mathbf{E}'\mathbf{S} \\ &= \mathsf{Frodo.Encode}(m) + \mathbf{S}'(\mathbf{A}\mathbf{S} + \mathbf{E}) + \mathbf{E}'' - \mathbf{S}'\mathbf{A}\mathbf{S} - \mathbf{E}'\mathbf{S} \; . \\ &= \mathsf{Frodo.Encode}(m) + \mathbf{S}'\mathbf{E} + \mathbf{E}'' - \mathbf{E}'\mathbf{S} \\ &= \mathsf{Frodo.Encode}(m) + \mathbf{E}''' \end{aligned} \tag{5.18}$$

It is a simple matter for $\mathsf{Frodo.Decode}(\cdot)$ (in Eq. (5.17)) to successfully return $m' = m$ if the combined noise $\mathbf{E}'''$ is small in comparison to the elements of $\mathsf{Frodo.Encode}(m)$. In fact, $\mathsf{Frodo.Encode}(\cdot)$ is constructed by "upscaling" values of m and subsequently, when decoding, using a rounding operation followed by a "downscaling", for each matrix element $k \in \mathbb{Z}_q$. The result is such that

$$\mathsf{Frodo.Decode}_{i,j}(\mathsf{Frodo.Encode}_{i,j}(k) + e) = k, \tag{5.19}$$

for each matrix position i, j and any error term e up to a certain limit, as determined by the FrodoKEM specification.

Care is taken when designing the parameters and how the various matrices are generated/sampled. For instance, the matrix A is sampled from uniform randomness over the whole range $\mathbb{Z}_q$ while the others in $\mathbf{E}'''$ use a more narrow gaussian distribution, centered over zero. The public parameters of the FrodoKEM scheme ensure a low probability of decoding errors since the elements of $\mathbf{S}$, $\mathbf{S}'$, $\mathbf{E}'$, $\mathbf{E}$ and $\mathbf{E}''$ are (probably) small and therefore the elements of $\mathbf{E}'''$ remains below the decoding limit. One can see from Eq. (5.18) that the matrix $\mathbf{A}$ acts as a way of masking the elements of $\mathsf{Frodo.Encode}(m)$ due to its multiplication with the secret matrix $\mathbf{S}$. Without knowledge of $\mathbf{S}$, it is difficult to remove the effects of $\mathbf{AS}$ from the ciphertext, and of course, it is a hard problem to recover $\mathbf{S}$ from $\mathbf{B} = \mathbf{AS} + \mathbf{E}$.

5.3 Kyber

Kyber is of Module-LWE design. So, it operates not on matrices of numbers in $\mathbb{Z}_q$, like FrodoKEM, but over matrices and vectors of polynomials in $\mathbb{F}_q[X]/\mathrm{f}(X)$. The main advantage is the size of the public key and the speed afforded due to the additional structure in the secret and public keys. The overall design of Kyber is similar to other (M/R)LWE designs, which is also the case when compared with FrodoKEM.

Key generation computes the the secret key **s** as a random vector of small[2]

[2] They have small coefficients; Roughly $-q/2 \ll a_i \ll q/2$, where a_i is the ith coefficient of the polynomial a.

polynomials. The public key is $(\mathbf{A}, \mathbf{t})$, where the matrix $\mathbf{A}$ is of small dimensions $(k \times k)$ with $k \in \{2, 3, 4\}$, depending on security level. $\mathbf{t}$ in this instance is calculated by

$$\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}. \tag{5.20}$$

The encryption function samples vectors $\mathbf{e}_1, \mathbf{e}_2, \mathbf{r}$ of random small polynomials, and then calculates $c = (\mathbf{u}, \mathbf{v})$ like so

$$\mathbf{u} = \mathbf{A}^T\mathbf{r} + \mathbf{e}_1 \tag{5.21}$$

and

$$\mathbf{v} = \mathbf{t}^T\mathbf{r} + \mathbf{e}_2 + m. \tag{5.22}$$

Before inserting the plaintext, it is first encoded by multiplication with a constant (i.e. upscaling), just like in FrodoKEM, so we get large coefficients.

To decrypt, Kyber uses the secret key in the following:

$$m' = \mathbf{v} - \mathbf{s}^T\mathbf{u}. \tag{5.23}$$

The final step then is to reverse the encoding of m' so that the original plaintext is reconstructed by rounding and downscaling.

The decryption works for the same reasons as it works for FrodoKEM, and thus the details are omitted here. It is enough to know that the combined error terms are, with very high probability, small enough to be discarded by the decoding, or decompression, as it is called in the case of Kyber.

The main speedup of Kyber vis-à-vis FrodoKEM is that the structure of Kyber enables the use of the Number Theoretic Transform (NTT) to perform translation of the problem input to the NTT domain, where polynomial multiplication is more efficient. NTT is an efficient algorithm, corresponding to the Discrete Fourier Transform (DFT), but over finite fields instead of the customary complex numbers. The NTT can be implemented using a variety of different finite fields, and the choice of field can affect the efficiency of the algorithm.

Chosen Ciphertext and Side-Channel Attacks

6.1 Chosen Ciphertext Attacks

A chosen ciphertext attack refers back to the security game defined in Section 2.5, where the attacker is given not only the power to construct whatever ciphertext he/she wishes but may also be given the decryption of any such ciphertext (except, of course, the specific challenge ciphertext given by the game). These attacks work because A) the scheme is *not*[1] proven to be IND-CCA secure or B) the scheme or implementation leaks some parts of its secret internal state. This latter case is not allowed in the IND-CCA model, recall Section 2.5, so it does not invalidate the security proof, but it shows that such proofs often make assumptions that do not always hold, in practice. In this section, a few different Chosen Ciphertext Attacks (CCA) will be discussed.

Decryption Failure Attacks (DFA) are sometimes also referred to as reaction attacks. And true to its name, such an attack considers the reaction of the target when performing decryptions (or decapsulations, depending on the setting). In short, a starting valid ciphertext c is generated. Then modifications $c' = c + e$ are introduced, first small ones and then progressively larger and larger, until a decryption failure is detected due to how the target reacts (perhaps by requesting a retransmission, or through a side-channel). Comparing the errors e causing a decryption failure with those who do not, the attacker gains some knowledge of either the message or the secret key.

Generally speaking, the authors of the first attack, presented in the following section, refer to the affected cryptosystems as closest-point cryptosystems. One may consider a closest-point cryptosystem as a subclass of schemes with malleability and the possibility of decryption failures. See the quote below:

[1] Or equivalently, there is a proof, but it is faulty.

> *Each of these systems could be considered a closest-point cryptosystem. That is, the ability to decode a ciphertext depends upon the ability to determine the closest "point" to the ciphertext in some linear space. For error-correcting code systems, this equates to the ability to determine the closest codeword in the linear space of codewords. For lattice-based systems, this equates to finding the closest point in a lattice. In all of these systems, one could consider the class of ciphertexts corresponding to a particular plaintext to be a sphere surrounding a point in the respective space (where the boundaries of the sphere are determined by an appropriate distance metric).*
>
> *By examining ciphertexts that are close to each other in the space (but possibly in different classes) we can determine the boundaries of this sphere and hence the center of the sphere. For some of the systems [...] the security of the system relies upon the inability to determine points in the respective space. [...]*
>
> *We feel that any [...] public-key cryptosystem with these properties will be vulnerable to the same sorts of attacks we present here.*

This prescient quote by Hall, Goldberg and Schneier in [HGS99] relates to the attacks presented in that paper. These include message recovery attacks against the code-based McEliece cryptosystem and key recovery attacks against the lattice-based Ajtai-Dwork PKC algorithm [AD97] and its Goldreich-Goldwasser-Halevi variant [GGH97]. Hall et al. showed that information about the message (in the case of McEliece) or secret key (for Ajtai-Dwork) can usually be determined, if the decryption fails. Leaving the attack on the Ajtai-Dwork PKC as a reading exercise for those interested, the attack on McEliece's scheme is briefly described in the following section.

6.1.1 DFA on McEliece

The attack on McEliece assumes one important property of the decoding algorithm selected by the implementation; The decoder will fail if presented with a vector with $t + 1$ or more errors. Recall, from Section 4.2, that t serves as the decoding limit of the underlying code. The assumption does not always hold, but in many cases it does, as shown by Hall et al.

If additional errors, recorded in the extra error vector $\mathbf{e}$, are introduced to the ciphertext $\mathbf{c}$ by simple modulo 2 bitwise addition, we get a modified ciphertext

$$\mathbf{c}' = \mathbf{c} \oplus \mathbf{e}. \tag{6.1}$$

The intuition behind the attack is this, if the extra error bits are introduced one at a time and if $\mathbf{c}'$ eventually results in a decryption failure we know, by the assumption above, that we have exactly $t+1$ errors. Thus, by finding all bits $i \in \mathcal{I}$ in $\mathbf{c}'$ which, if individually flipped, causes the decoding to once again succeed, we gained the knowledge that those bits must be in error in $\mathbf{c}'$.

This knowledge is enough to completely remove the internal error vector $\mathbf{z}$ from $\mathbf{c}$ to get the corrected ciphertext $\mathbf{c}_\mathbf{c}$. Without the error the original message can be recreated by applying the technique from [AM87] which only requires the selection of k error-free positions of $\mathbf{c}$. The selection of k error free positions allows the attacker to solve for $\mathbf{m}$ by setting up the equation

$$\mathbf{c}_\mathbf{c}' = \mathbf{m}\mathbf{G}', \tag{6.2}$$

where $\mathbf{c}_\mathbf{c}'$ is the selected k positions of $\mathbf{c}_\mathbf{c}$ and $\mathbf{G}'$ is the same k selection of columns from $\mathbf{G}$.

6.1.2 Bleichenbacher attack on RSA

The term reaction attack might, to the author's best knowledge, be traced to the above-cited paper by Hall et al. However, a year prior the Bleichenbacher attack [Ble98] was published. This attack made use of a so-called padding oracle for RSA using the PKCS#1 v1.5 padding scheme. It was made possible by error messages explicitly letting users know if the padding of the decrypted messages was correct, or not. This knowledge could be leveraged by multiplying the encrypted message with a selected value s such that:

$$c' = cs^e \mod N. \tag{6.3}$$

Sending c' to the decryption oracle one gets

$$m' = (c(s^e))^d = c^d s^{ed} = ms \mod N. \tag{6.4}$$

The property of RSA showcased above is called the homomorphic property. It states that some operations on the ciphertext propagate in a predictable way to the plaintext, even if the plaintext is unknown. This property is what enables the wholly separate and quite interesting research field of homomorphic encryption, which allows servers to make calculations on encrypted and unknown data, though it will not be discussed more in this dissertation.

When the receiver reads m' it will most likely not adhere to the padding scheme of PKCS#1 v1.5. The fault of many early implementations was to respond with an error message, effectively providing a decryption failure oracle of the first two bytes, which must be 00 and 02, according to the padding scheme.

Trying many choices of s we eventually arrive at one that returns the specific error message we are looking for. Then we have gained the following information:

$$2B \leq ms < 3B \mod N, \tag{6.5}$$

where $B = 2^{8(k-2)}$ of which k is the number of bytes of N. Recall, that the first two bytes are known (here interpreted as the most significant digit, two) and thus ms must be represented by the $k-2$ remaining bytes of the message.

By collecting many valid choices of s we may eventually know enough to recreate m in its entirety.

6.1.3 DFA on NTRU

Following the publication of [HGS99], Hoffstein and Silverman realized in [HS99] that a similar reaction attack on lattice-based NTRU was possible. The attack on NTRU follows a similar structure as the one against both the McEliece and the Ajtai-Dwork schemes and it is described briefly below.

NTRU encryption, see Section 5.1 and Eq. (5.4), is of the form

$$\mathrm{c} = \mathrm{rh} + \mathrm{m} \mod q \tag{6.6}$$

which, due to Eq. (5.10), informs on the smallest possible modification c'. For any choice of $0 \leq i \leq N$ a modification in the form of

$$\mathrm{c}' = \mathrm{c} + npX^i \tag{6.7}$$

is likely to successfully decrypt for small choices of $n > 0$. Conversely, if f has a matching $+X^i$ term and the intermediate decryption polynomial a has the corresponding coefficient within np of the upper bound, of q relation, the assumption of small coefficients no longer holds. This will cause a decryption failure. Analogously, some negative choices of n informs on $-X^i$ terms of f. How to reconstruct the secret key from the above information is not shown here, it is a process of shifting and duplicating the secret coefficients as well as collecting from multiple messages. It is enough for the above to show why the attack leaks some information about the secret key. For more details, see [HS99].

A mix of reaction and chosen ciphertext attacks[2] on NTRU are [JJ00; Hon+02; How+03; GN07].

6.1.4 CCA on (M/R)-LWE-based cryptosystem

An attack on RLWE-based key exchange schemes under the static key paradigm was first reported on by Fluhrer [Flu16]. In this setting, a user is employing a, supposedly, Diffie-Hellman (DH) key exchange drop-in replacement. The idea was to, like with DH, allow a server to make use of a static key-share while connecting clients make new, ephemeral, key-shares. Fluhrer showed how RLWE schemes can be broken in such static-ephemeral settings. The fully ephemeral setting, with fresh key shares for all parties, is not affected by this attack.

[2] Chosen ciphertext attacks differs from reaction attacks in that the result of the decryption is required to mount the attack.

The attack makes use of a key mismatch oracle, to gain information on if a guessed coefficient of the secret polynomial is correct or not. If the guess is correct the key agreement is successful and the generated symmetric key matches for both parties (this can be tested by sending an encrypted message and waiting for a response). Since the target key-share is static the attacker may submit multiple guesses and then proceed to the next coefficient once a good guess is confirmed by the oracle. Those kinds of attacks are also known as misuse attacks since the attack scenario is the improper reusing of ephemeral-only keys. In other words, they take advantage of only IND-CPA secure schemes in settings that require IND-CCA security.

The key mismatch oracle is related to decryption failure oracles in that it tells $m' \overset{?}{=} m$ for both key exchange algorithms and KEMs, whereas for PKC and KEM systems, a decryption failure oracle informs on the same equality but with the distinction that introduced errors/noise in the ciphertext sometimes results in m' with no relation to m due to decoding or correction failures.

Key recovery attacks on both Kyber and FrodoKEM are possible if a decryption failure can be induced and detected [Rav+20]. In the following, we briefly describe the attack on FrodoKEM[3], although the attack on Kyber is similar. The attack, at its core, retrieves a single element $\mathbf{S}[0][0]$ of the secret matrix. During encapsulation, the first element of the matrixes $\mathbf{B}'$ and $\mathbf{C}$ are carefully selected with values $\mathbf{B}'[0][0] = k_{\mathbf{B}'}$ and $\mathbf{C}[0][0] = k_{\mathbf{C}}$ and all other elements set to 0.

Briefly recalling the decryption procedure of FrodoKEM, we have

$$\begin{aligned} \mathbf{M} &= \mathbf{C} - \mathbf{B}'\mathbf{S} \\ m' &= \mathsf{Frodo.Decode}(\mathbf{M})\text{'} \end{aligned} \tag{6.8}$$

where m' is decoded by $\mathsf{Frodo.Decode}$ as a scalar value. Though it is interpreted as a bitstring when values of $\mathbf{B}'[i][j]$ are mapped to specific bits, here given in the notation $m_{i,j}$. The values of $m_{i,j}$ are given below:

$$m_{i,j} = \begin{cases} \mathsf{Frodo.Decode}(k_{\mathbf{C}} - k_{\mathbf{B}'}\mathbf{S}[0][0]), & \text{if } i = 0, j = 0 \\ \mathsf{Frodo.Decode}(-1 \cdot k_{\mathbf{B}'}\mathbf{S}[0][j]), & \text{if } i = 0, j \neq 0\ . \\ 0, & \text{otherwise} \end{cases} \tag{6.9}$$

It is possible to select the values of $k_{\mathbf{B}'}$ and $k_{\mathbf{C}}$ such that $m_{i=0,j=0}$ depends only on $\mathbf{S}[0][0]$ and $m_{i,j} = 0$ for all other values of i, j. Adding a further restriction, that the decrypted message should take the either value of $m' = 0$ or $m' = 1$ we obtain a Plaintext-Checking oracle (PC oracle). This can be done since secret coefficients in FrodoKEM (and Kyber) are given within a range that depends on

[3] It differs somewhat from the attacks presented in papers III and VI, hence it is included here.

q. As such, some values of $\mathbf{S}[0][0]$ results in a message bit of zero and others result in a message bit of one.

By submitting a number of guesses with different ciphertexts to the PC oracle it is possible to find out the value of $\mathbf{S}[0][0]$. The other positions of $\mathbf{S}$ can be obtained in a similar fashion by exchanging the zero positions of $\mathbf{B}'$ and $\mathbf{C}$. The details of key reconstruction are omitted from this description, see [Rav+20] for more information.

6.1.5 DFA on QC-MDPC based cryptosystems

In 2016, Guo, Johansson and Stankovski showed how to mount a key-recovery attack on the QC-MDPC cryptosystem [GJS16; GJW18]. The attack on QC-MDPC, being the antecedent system to the BIKE cryptosystem (see Section 4.3), is highly relevant to this dissertation due to it being partway the foundation of papers I and V, as well as being the motivation behind paper IV.

Guo et al. showed how the small probability of decoding failures can be exploited to reveal dependencies on the secret key. The aim is to, given only the public-key generator matrix $\mathbf{G}$, recover $\mathbf{h}_0$ and thus be able to reconstruct the secret parity check matrix $\mathbf{H}$ via the first cyclic sub-matrix $\mathbf{H}_0$ for which $\mathbf{h}_0$ is the first row.

By examining the decoding procedure of different error patterns the authors discovered a correlation between the Decoding Failure Rate (DFR) and matchings of cyclic pair-wise distances of ones (1's) in both $\mathbf{h}_0$ and the generated error pattern. In short, if the distance between any two non-zero elements of $\mathbf{h}_0$ is d then, shown empirically, error patterns also containing the same pair-wise cyclic distance d is measurably more difficult to decode.

The set of distances between any two non-zero elements in a vector is called the distance spectrum. By collecting the distance spectrum of many error patterns which fail to decode, the most common distances showed a very strong correlation with the distance spectrum of the secret key. The secret key can easily be reconstituted from the aggregated distance spectrum of the error patterns.

This attack is very strong in the IND-CPA setting, where the attacker is able to determine the error pattern, and even construct more difficult error patterns by increasing its weight. In the IND-CCA setting the process is more expensive due to no longer being able to choose the error pattern and thus must rely both on "natural" decoding failures and on some way to distinguish them.

6.2 Side-Channel Attacks

So far it has not been discussed *how* the target can be observed to make the above attacks possible. In the case of IND-CCA secure PKC schemes, such information is not, theoretically, available. By using side-channel information we might however be able to mount a Side-Channel Attack (SCA) to gain exactly the insight required to implement the various oracles necessary for the attacks.

> *Essentially, all models are wrong, but some are useful.*
>
> — George Box

Security proofs, by their very nature, can only consider side-effects of operations, if they are explicitly included in the model. Being a challenging proposition, security proofs typically do not model the leakage of internal states. In the following, we only consider a leakage if it relates to, or depends on, non-public input, in some way. Leakages that depend solely on public information (e.g. the public key) can by their nature not give-away any further useful information.

A physical (i.e. hardware- or software-) implementation of a cryptographic system can be leaky by virtue of timing variations [BB03; Str10; BT11; Str13; Bru+16; Kau+16; DAn+19; Waf+19; PT19], power consumption [KJJ99; Ngo+21; Ham+21; GJJ22; Sch+22a], electromagnetic emanations [Rav+20; GLG22], acoustic signals [GST14; GST17] etc. The list goes on.

Fault injection attacks [Bar+12], which are also known as active side-channel attacks, is another interesting SCA-related attack vector. In such attacks, a target system is affected by an external stimulus such that the targeted function misbehaves, in some manner. The critical insight is highly dependent on the targeted system and the fault that is introduced. The commonality lies in the idea that by comparing the output of correct encryption/decryption with the output of faulty encryptions or decryptions the attacker can gain some crucial information on the private inner state of the algorithm. Usually, this is used to attack protected hardware in an adversarial setting, such as smart-cards.

There are many techniques for extracting information from leakages. Such techniques include simple power analysis [KJJ99], differential power analysis [KJJ99], template attacks [CRR03], correlation power analysis [BCO04] and test vector leakage assessment [SM15]. Also, artificial intelligence techniques such as machine learning and deep learning have increased in recent years.

The focus of this dissertation is towards DFAs against PQC schemes, aided by SCA. There are software-based timing attacks in papers III and V and a simple power analysis attack in paper VI. Therefore this section is not intended as a full primer on more advanced SCA techniques; Instead, we refer the reader to the respective paper where each technique is introduced. That being said, some related works are briefly introduced below, related to the aforementioned timing attacks.

6.2.1 Timing Attacks

Timing attacks against PKC algorithms were first introduced by Kocher in his seminal work [Koc96] "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". Consider, for example, the RSA decryption operation

$$m = c^d \pmod{N}, \tag{6.10}$$

where the decryption exponent d is part of the secret key. It was shown that, for instance, the modular exponentiation above was often implemented by the "square-and-multiply" algorithm which by timing data leaks bits of the exponent. In particular, the algorithm is usually implemented such that, for each iteration i of the algorithm, it checks if the exponent has the particular bit d_i set ('1') in the secret exponent. If that is the case, a conditional multiplication is executed, otherwise not. Kocher showed how this information could be leveraged to obtain the secret exponent d by passively observing multiple ciphertexts and timing measurement pairs. By comparison against a model, each bit of d can be guessed, one at a time until the whole secret key is reconstructed.

Later, Brumley and Boneh showed that "Remote Timing Attacks Are Practical" [BB03]. In this work, they showed that it is actually feasible to attack servers running the, then, latest version of OpenSSL in a close-to-real-world scenario. This was previously believed to be very difficult due to the inherent noisy environment of a general-purpose network-connected server with lots of traffic.

In [Str10; Str13] Strenzke showed key-recovery timing attacks against the McEliece decryption operation. Here the secret permutation and the syndrome inversion steps were the leaky operations and by constructing invalid ciphertext with a specific error weight it was possible to distinguish the number of iterations performed by an inner part of the error correction algorithm. It turned out that this number was directly correlated to a specific coefficient in the secret error location polynomial, used by the error correction algorithm. In the end, it is possible to build a list of linear equations which describe the secret permutation. With enough data, the list of linear equations can be solved by the Gaussian elimination algorithm.

Later, Brumley and Tuveri showed that "Remote Timing Attacks Are Still Practical" [BT11]. This time Brumley et al. show a timing attack against OpenSSL's Montgomery ladder implementation for elliptic curves. The authors managed to mount a lattice attack that recovered the private key of a TLS server, authenticated by ECDSA signatures.

D'Anvers et al. [DAn+19] made a timing attack on PQC schemes that utilize error-correcting codes to reduce the decoding failure rate. They used timing information to distinguish between ciphertexts that result in an error before decoding and ciphertexts that do not contain errors. Of course, this works only if the ECC decoding algorithm has a variable execution time, which was the case for at least the NIST PQC submissions LAC and Ramstake. The reason for implementing it that way was presumably that the algorithm authors did not consider that the above distinction counted as secret input.

Wafo-Tapa et al. [Waf+19] and Thales Bandiera et al. [PT19] both showed how to exploit an earlier version of the HQC scheme due to a correlation between the weight of the error to be decoded and the running time of the non-constant time BCH decoding algorithm (now replaced by an RMRS code and decoder).

6.2.2 Cache Timing Attacks

In [Bru+16] Bruinderink et al. present the first side-channel attack on a lattice-based signature scheme, using a cache-attack technique (see below). The scheme in question was the Bimodal LattIce Signature Schemes (BLISS) [Duc+13] and the target was the discrete Gaussian sampler. The sampler was used to construct a blinding (or noise) polynomial in order to make the signature statistically independent of the secret key. If an attacker learns the noise polynomial for a number of signatures it is possible to reconstruct the secret key by guessing and linear algebra.

The technique used was the Flush+Reload cache attack. This is an active way of exploiting timing side-channels that would not otherwise be measurable. It requires local code execution privileges in order to reveal secret dependent memory access by inducing and measuring timing variations of neighboring processes.

The processor cache in your computer is a collection (a bank) of memory cells located inside the processor, used to bridge the speed gap between the fast processor and the slow primary memory. Several cache layers are typically used, such as L1, L2 and L3, where L1 is the smallest, fastest and closest to the execution core. In short, whenever the processor performs memory accesses it looks through the local caches first and uses the result whenever a match is found. If a match is not found in any of the caches, then an expensive read operation is performed on the main memory bank. Due to the small size, strict eviction policies of the processor determine what data is kept or replaced in the caches. These policies are hard-coded by the processor manufacturer. Caches are shared between processes and thus one process may cause an eviction of another process's data by exploiting the cache eviction policy of the processor. A cache attack makes use of the fact that when an attacker uses the same cache as a victim, victim memory accesses change the state of the cache. The attacker can use the measured timing variations to check which memory blocks are cached and from that deduce which memory addresses the victim has accessed.

In [Bru+16] Bruinderink et al. describes the Flush+Reload attack:

> *A Flush+Reload attack uses the* `clflush` *instruction of the x86-64 architecture to evict a memory block from the cache. The attacker then lets the victim execute before measuring the time to access the memory. If during its execution the victim has accessed an address within the block, the block will be cached and the attacker's access will be fast. If, however, the victim has not accessed the block, the attacker will reload the block from memory, and the access will take much longer. Thus, the attacker learns whether the victim accessed the memory block during its execution.*

6.2.3 Constant time implementations

By this time it is clear, and indeed well-known, that cryptographic algorithms must be implemented in constant time. In cryptography, ”constant time” refers to an algorithm or implementation that takes the same amount of time to execute regardless of the input data. However, as per the discussion given in the introductory paragraphs of Section 6.2, one may relax this constraint slightly. Instead, one may be implicitly talking about constant time only in relation to *secret elements* of the input data. To achieve a constant time implementation we must consider the following operations [Por17]:

Memory accesses As we have seen above, it is entirely within the realm of possibility for an attacker to glean insight into the spatial and temporal locality of memory accesses, primarily through cache-timing attacks. Therefore, it is important to make sure memory *addresses* do not have any dependency on a secret element.

The most immediate consequence is that common implementation techniques such as lookup tables cannot be used unless they are small enough to fit within a single cache-line[4].

Conditional jumps The above point on memory access apply also to program execution flow; Each processor instruction resides in memory and thus a branching in the execution flow leads to a memory read of the location of the taken program branch, to find the next instruction for the processor to execute.

However, it is even worse than that. Today, almost all processors, except maybe for those intended for constrained environments, have a branch prediction feature, where the processor guesses which branch will be taken, before the branch condition is executed. This allows the processor to pre-fetch the next instruction and for instruction pipelining[5], considerably improving performance. If a miss-prediction happens, a detectable delay occurs since the processor must discard whatever work was performed in advance due to its (erroneous) branching guess. Branch, or jump, prediction uses both static rules and a dedicated cache system, which can lead to a similar kind of attack as against table lookups.

Naturally, this is a problem only if the condition is secret. For instance, when implementing AES-128, there are ten rounds, so an implementation may use a loop with a conditional jump that will exit after the tenth round.

[4] A cache is divided into cache-lines, which is the smallest individually addressable unit of the cache. One common cache-line size is 64 bytes, for the L1 cache, closest to the processor.

[5] A performance enhancing technique where processor instructions are broken down into smaller parts and executed in several steps. Thus freeing up resources for handling the first step of the next instruction before being finished with the next step of the current instruction.

That AES-128 includes ten rounds is not secret, so that specific conditional jump is not problematic.

Integer divisions Some processor architectures are implemented such that integer divisions have two code paths, one for all possible inputs and one which is optimized for small divisors or dividends. Thus the speed of the integer division instruction might reveal the size of the inputs. The same problem might even occur on architectures without hardware support for integer division, in this case, the program compiler might supply one of its own built-in subroutines, with the same input-size dependent optimization. Note that this problem may occur for both the division and modulo operations ('/' and '%' operators in the C programming language).

Some divisions might be optimized into bit-shifting and bit-masking operations, depending on the inputs (power of 2 divisions are a trivial example). However, it can sometimes be complicated to predict the real effects of such operations, since the C-compiler can make surprising choices. For instance, if signed types are involved, divisions by powers of two may still involve some special code with conditional jumps because the C standard mandates that (-1) / 2 == 0, while most CPUs will right-shift -1 into -1 (arithmetic shift with sign extension).

Shifts and rotations Processors without hardware support for constant time bit-shifting operations can leak shift and rotation counts. This is mostly a legacy problem since almost all modern processors feature a so-called "barrel shifter", which is constant time.

Multiplications Another, mostly legacy, problem to watch out for is multiplications, which on most modern processor architectures are implemented in constant time. However, some older processors use special code paths for small inputs, which are faster.

Sometimes even adhering, or rather believing oneself to be adhering, to the above guidelines might not be enough. Kaufman et al. [Kau+16] showed how a particular implementation, of Elliptic Curve Diffie Hellman key exchange based on the curve X25519, could still be subject to a timing attack, even though adhering to strict implementation security guidelines [LHT16]. In this case, the culprit was in a platform-dependent runtime library that implemented integer multiplication in an input-size-dependent manner. The integer multiplication was performed by a constant time Montgomery ladder implementation given by [LHT16], which was believed to be secure, and indeed it was, on *most* platforms.

The paper [Kau+16], is quoted below:

> *Once a security design is implemented, whatever effort is put into protecting each part of the code, there still remains a strong possibility of a*

timing leak. It is virtually impossible *to have control over all the parameters at stake. Compiler and processor optimizations, processor specificities, hardware construction, and runtime libraries are all examples of elements that cannot be predicted when implemented at a high level.*

6.3 Classification of Oracles

So far, a number of different attacks have been presented, many of these use different kinds of oracles, many based on side-channel information. This section makes an attempt to gather and categorize them. To be clear, IND-CCA secure schemes nominally prevent this kind of oracles, (see the discussions the FO transform and on the CCAs in Sections 2.6 and 6.1). However, as has already been discussed, undesired side-channel information may still allow these oracles to exist, in practice. The work of Ravi, Chattopadhyay, D'Anvers and Baksi [Rav+22] is used as basis for the following listing:

Plaintext-Checking oracle (PC oracle) The key-recovery attack given by [Rav+20], described briefly in Section 6.1.4 is an example of an attack utilizing a plaintext checking oracle.

That is, a special low-weight ciphertext is generated, where the decrypted plaintext depends on only a single coefficient of the secret key. One of two cases may be observed in the decrypted message; It is either $m' = 0$ or $m' = 1$. The plaintext oracle in other words checks for which plaintext is given during decryption.

The PC oracle can be either binary [DAn+19; Rav+20; Uen+22], as described above, or parallel [Raj+22; Tan+22]. In the parallel PC oracle, the oracle provides more than a single bit of information per query. This is accomplished by generating a ciphertext where multiple bits of the decrypted message depend on different solitary coefficients of the secret key. The parallel PC oracles in the cited works were realized via power/EM-side-channels.

Decryption Failure oracle (DF oracle) DF-based oracles usually start from a valid ciphertext, which with a very high probability successfully decrypts. Then the ciphertext is modified bit by bit until a decryption error is detected by the DF oracle. Once detected the attacker uses the knowledge of decoding/decrypting limits of the cipher scheme, the encrypted plaintext and information of the performed ciphertext modifications, to gain exact insight into the secret key.

The attacks presented in Sections 6.1.3 and 6.1.5 are examples of attacks enabled by DF oracles, as is the power/EM attack in [Bha+21].

Full-Decryption oracle (FD oracle) The final oracle discussed in this dissertation is the FD oracle. This oracle is able to provide not only a single or a few bits of information per query, but the entire message. Xu et al. [Xu+22] showed with an EM/power attack that it is possible to construct ciphertexts such that all message bits correspond to a unique secret key coefficient.

The FD oracle is simply a special case of the parallel PC oracle where all the bits of the message are utilized fully, which is only partially the case for the parallel PC oracle. Examples of FD oracle based attacks are [Xu+22; Rav+21; Ngo+21; NDJ21; WND22a; WND22b].

Key mismatch oracles (for misuse attacks) are very similar to both binary PC oracles and DF oracles, although the key mismatch concept is primarily used for considering the robustness of only IND-CPA secure schemes when improperly reusing ephemeral keys. Since they are used in different settings and for differing purposes, this oracle type is not included in the list above.

Decryption failure attacks, as enabled by DF oracles, are the main theme of this dissertation, as these 1-bit information oracles are the ones that most easily lend themselves to software-based timing attacks, which has been the main research goal of this Ph.D. project. This will be discussed more in the following chapter.

Contributions and Conclusions

This thesis aims to explain how the concept of Decryption failure attacks was further developed during Alexander's research project and how it was used to identify such weaknesses in relevant next-generation PQC encryption schemes. Only by identifying such weaknesses early and by attempting to exploit them can the ramifications be known and hopefully be prevented from impacting real-world use cases.

7.1 Contributions

This dissertation focuses on the intersection of cryptographic software implementation issues, cryptanalysis and coding theory with some sprinkles of lattice cryptography on top. Let us summarize the different areas of the included papers below, an illustration is also available in Fig. 7.1.

Crypto implementation Papers III to VI. Of these papers III, V and VI focuses on discovered issues, or side-channel leakages, in implementations of cryptographic software. Paper IV focuses on an alternative implementation and improvement of a specific part of a KEM algorithm.

Cryptanalysis Papers I to III, V and VI. All but one of the included papers attempt to improve the state of the art regarding the cryptanalysis of existing schemes. Three papers (III, V and VI[1]) find novel source-code vulnerabilities, two papers (I and VI) improve upon existing attacks with new techniques and one paper (II) gives theoretical cryptanalytical results regarding the practical security level of proposed parameters.

Coding theory Papers I and IV to VI. These papers relate to coding theory either as an analytical tool (paper VI) or by virtue of relating to the code-based

[1] Though, the focus is on the new SCA framework, not the power attack.

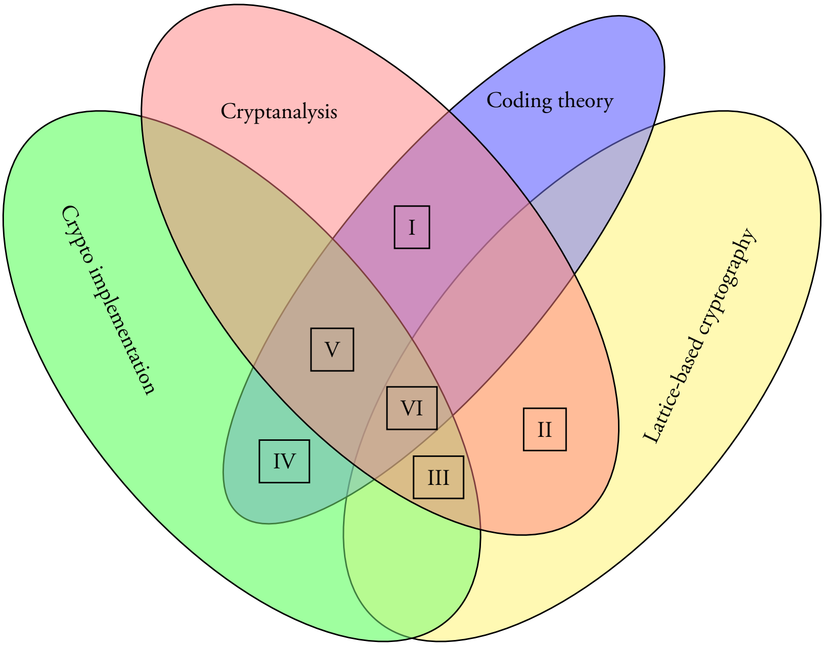


Figure 7.1: Venn diagram of the included papers' contributions.

KEM schemes BIKE (QC-MDPC) and HQC. Notably, McEliece is absent from this list.

Lattice-based cryptography Papers II, III and VI. Lattice-based KEM schemes is an important sub-topic of this dissertation and the KEM schemes NTRU, Kyber and Frodo are subject to various level of scrutiny in these papers.

7.1.1 Paper I: Error Amplification in Code-based Cryptography

This paper expands upon the original "GJS" attack [GJS16; GJW18], see Section 6.1.5, such that IND-CPA secure QC-MDPC based schemes can be attacked with far fewer oracle queries. The contributions are primarily two-fold.

First, it was explained that a measure of closeness can be exploited for "difficult to decode" error patterns, such that if one such pattern is obtained then small modifications to this pattern can with high probability yield new error patterns with the same properties. Due to the similarities of these patterns, the whole pattern is not considered in the attack, but rather the differences in the distance spectrum between pattern e_i and e_{i+1} are considered. In this manner a chain of error patterns was generated, each pattern being a single modification of its immediate ancestor. Here a modification refers to the move operation of a set bit into a new unoccupied position in the error pattern. Basically, the algorithm starts from a “genesis” pattern that it randomly modifies until it finds a single modification that also results in a decoding failure. Even better, it was shown that even if the next candidate pattern e^*_{i+1} does not actually cause a decoding failure, that is still information that can be incorporated to improve the attack. Simulations showed a massive increase in the decoding failure rate of the scheme under attack.

The second major contribution was that the presence of side-channel information in the decoder could easily be used to improve the attack and even the act of finding the genesis pattern. Consider modifications resulting in a pattern that goes from decoding failure to decoding success. For these one can additionally record, due to timing information from the decoder, a measure of decoding difficulty. I.e. if a pattern modification was determined to still be “difficult” to decode it could be counted in the same manner as a decoding failure in the post-processing step of the attack.

Even more important though, the process of finding the genesis pattern could be considerably improved by the side-channel information simply by applying the chaining method again. The idea was that instead of relying on decoding failures, we rely instead on a measure of how difficult a pattern is to decode, in relation to a previous pattern. So, starting from a random pattern, loop through all singular modifications until a new pattern is found that is “more difficult” to decode and use that pattern as the new base pattern to modify. Repeat until a decoding failure is found.

The simulations were implemented in the C programming language.

To summarize this paper, the research community’s awareness related to the importance of constant time implementations was further reinforced. Also of import is the knowledge of how the decoding failure rate can be artificially increased by an attacker against only IND-CPA secure schemes. The behavior of the decoders was simulated while subject to especially hard-to-decode error patterns and it could be seen that different decoders were affected differently. Knowledge of how different decoders behave is important for DFR analysis and could potentially affect the static-key security analysis of the QC-MDPC-derived cryptosystem BIKE, which as of this writing, remains a candidate in the NIST PQC standardization effort.

Though not discussed in the paper, the techniques could be used in an IND-CCA setting when paired with SCAs, as evidenced by paper V.

7.1.2 Paper II: Decryption Failure Attacks on IND-CCA Secure Lattice-Based Schemes

This paper is the result of a merge of [DVV18] and [GJN19]. The contributions lie primarily in the cryptanalysis investigation related to the included KEM schemes’ decryption failures. Specifically, it contains a study, for each scheme, on the amount of information that is leaked per recorded decryption failure. A technique, called failure boosting, details how decryption failures might be gathered more quickly than the random decryption failure rate would otherwise suggest. It works by brute-forcing messages which through the FO transform are converted to a seed that has some small probability of generating a "weak" ciphertext, according to the criteria listed in the paper. Depending on if the targeted scheme incorporates multi-target protection or not, the weak ciphertexts are independent of the used keys and thus failure boosting may be considered a global one-time computational trade-off. For schemes with multi-target protection (e.g. Kyber) failure boosting is useful as a computational trade-off if the attacker has access to A) a limited number of calls to the decryption oracles or B) if the attacker has access to a quantum computer with Grover's algorithm to speed up the search. A theoretical framework and a python implementation are provided to calculate the amount of effort required to find one ciphertext that triggers a decryption failure.

A statistical model was developed on the leakage by which the residual entropy of the secret can be estimated after a certain number of failures is collected. The estimate of the secret can be used to construct an easier problem that can be solved faster and the effects on the security level of the targeted scheme could be calculated. The attacker could significantly reduce the security of some schemes. But for the targeted schemes, the number of decryption queries required was above practical limits.

The paper then proceeded to develop a generic weak-key model which exploits the fact that some ciphertexts have relations to the secret key, which makes the ciphertext-secret key pair exhibit a higher decryption failure rate. The attack consists of a pre-computation phase where messages and corresponding error vectors are filtered and gathered. This is followed by a query to the decryption oracle, for each message. In the final post-processing step, the few resulting decryption errors are gathered and analyzed to reveal the secret key. The attack model is applied to the NIST PQC submission ss-ntru-pke. A Rust implementation of parts of the attack is provided.

This paper is well-cited and has had some influence on NIST PQC scheme parameter selection.

7.1.3 Paper III: A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM

This paper firmly establishes the importance of regarding the results of the re-encryption and ciphertext comparison steps of the FO transform as secret information. As such, these steps must be performed in constant time, which was not the case for a number of different NIST PQC submission implementations. The generic attack in question was a simple decryption failure attack against any scheme which utilize the FO transform and could be categorized as "closest-point cryptosystems", (see Section 6.1), which is primarily lattice-based or code-based schemes. The attack assumed that the implementers of the targeted KEM scheme did not consider the re-encrypted ciphertext as secret information, and thus did not offer any constant time protections for the ciphertext comparison step. Considering that the original ciphertext is indeed considered public information, it was an easy mistake to make.

The attack was implemented, in Rust, and simulated against the otherwise constant time-protected reference implementation of FrodoKEM, duplicated into the open-source library provided by the Open Quantum Safe project. This paper directly influences the implementation of the targeted scheme and thus can be said to have had a real-world impact.

7.1.4 Paper IV: A Weighted Bit Flipping Decoder for QC-MDPC-based Cryptosystems

The contributions in this paper are given by a new "Weighted Bit-flipping" (WBF) iterative decoder which would be a suitable alternative to the BGF decoder in the BIKE submission to the NIST PQC standardization process.

This paper's relevance to the current dissertation's topic lies not in the paper's contents but in its motivation. That is, research into ways of lowering the DFR of code-based KEMs is relevant for defending against DFAs.

It is explicitly stated in the BIKE specification that implementers may freely choose between decoders as long as the resulting decoding failure rate is low enough. For the ephemeral-key use case (recommended by BIKE) it must simply be low enough to not hinder practical integration into higher-level cryptographic protocols. In the case of static, or reusable, keys BIKE is IND-CCA secure on the condition that the DFR is at least as low as $2^{-\lambda}$, where λ is the targeted security level.

While the static-key use-case is not recommended by the BIKE specification it was shown in the paper that the simulated DFR of the new WBF decoder, when extrapolated, indeed appears lower than that of the BGF decoder, which already *appears* to be sufficient for the static-key use-case. To combat the extra iterations required (and thus increased workload) by the WBF decoder a hybrid decoder was introduced which combined only two iterations of WBF, followed by the BGF decoder with a reduced number of iterations.

This hybrid decoder showed better DFR than the original BGF decoder, while not adding as much computational overhead as the pure WBF implementation.

7.1.5 Paper V: Don't Reject This: Key-Recovery Timing Attacks Due to Rejection-Sampling in HQC and BIKE

This paper is the result of a pre-publication merge, with [HLS21], due to the independent discovery of the same weakness. Here the target is the, supposedly, constant time-protected software implementations of the BIKE and HQC KEM schemes. Specifically, it targets the variable-time rejection sampling algorithm. This paper explains how the dependency on the decrypted plaintext is in fact exploitable as a key recovery attack via a decryption failure oracle.

Rejection sampling is an efficient algorithm for generating binary vectors of a fixed weight w. Simply put, it samples w positions p_i where $0 \leq i < w$ of an all-zero vector. For each duplicate position $p_i = p_j$, where $0 \leq j < i$, the sample is rejected and a new sampling is attempted, subject to the same condition. This means that the number of rejections, and thus, resamplings required, depends on the seed given to the randomness sampling algorithm, (also known as a pseudorandom number generator). In the case of HQC and BIKE, the seed is (the hash of) the plaintext during encapsulation. During decapsulation, the seed is (the hash of) the decrypted plaintext, before being given to the re-encryption step.

If a decoding error occurs in HQC or BIKE, the decrypted plaintext is different and thus the seed differs. Subject to a different seed, the rejection sampling algorithm (probably) requires a different number of rejections to complete. This is enough to construct a timing-based decryption failure oracle and mount an attack.

The paper is divided into the parts specified in the following. A description of the generic problem, the specifics on the HQC vulnerability and the same for the BIKE vulnerability are of course in the paper. How to leverage the described attack together with the techniques from paper I to attack BIKE, despite an estimated DFR that corresponds to the security level of the scheme is an important contribution. For some versions of the BIKE implementation an additional check, not given in the specification, effectively stopped the attack on BIKE. This additional check, however, opened up an even more efficient message recovery attack, as detailed in the paper. The attack on both schemes was implemented and evaluated, indicating the practical applicability of the attacks. Some possible countermeasures applicable to both schemes, implemented and tested on HQC, rounds off the paper.

This paper directly influences both the implementations and the specifications of the targeted schemes and thus can be said to have had a real-world impact [Sen21; DGK23].

7.1.6 Paper VI: SCA-LDPC: A Code-Based Framework for Key-Recovery Side-Channel Attacks on Post-Quantum Encryption Schemes

This paper introduces a new framework for performing side-channel assisted DFAs and other CCAs with a reduced number of oracle calls. By utilizing coding theory, and specifically LDPC codes, the framework is practically demonstrated to considerably improve the attack complexity of two wildly different scenarios, highlighting the applicability of the new method.

The first scenario is a novel power attack on Kyber against the latest known masked open-source implementation. The attack, shows a considerable improvement over the state of the art, even though the target implementation is a new protected (masked) version of Kyber-768 from the open-source mkm4 library, running on a 32-bit ARM Cortex-M4 CPU.

The second scenario is against HQC where the LPDC code is employed in a source compression coding mode. Due to the low amount of entropy in the noise vector, this technique enabled reliable and automatically corrected recovery of all 17669 positions of the noise vector with only about 10000 idealized timing-based, binary, DF oracle calls. The vulnerability used for demonstration purposes was the timing-based pure software attack on HQC from paper V.

The framework establishes relationships between several positions/coefficients of secret elements which it can then use with coding techniques in order to either correct imperfect oracle calls, or like in the attack on HQC use for source coding purposes. In both cases, the number of required oracle calls is considerably reduced.

As of this writing, this paper has not yet had the opportunity to make an impact on the research community. By pure speculation, however, it is not far-fetched to believe the techniques presented here will be successfully reproduced by others and used to make future side-channel assisted attacks far more efficient.

7.2 Topic relevance

The great truism of cryptanalysis “Attacks always get better, they never get worse” applies especially well to papers I and VI. This is an important research topic in itself because the efficiency of attacks often serves as a measuring stick of how much effort should be spent on mitigation and prevention. Novel attacks and/or leakages were the focus of papers III, V and VI and such papers fill an even more important role in the cryptanalytical research space. After all, design or implementation issues can only be fixed if they are known. Providing alternative sub-components, as in the case of paper IV, of ”maybe-to-be-standardized” schemes is one of the things that drive forward innovation and questions the status quo, even if the alternative solutions are not selected for inclusion. In the case of paper II the contributions lie more towards the theoretical nature of cryptanalysis and, while useful in and of itself, heightens the knowledge and confidence of the research community in a more general way. Especially in the ever-important regard of what security levels certain schemes fulfill, and the confidence of those estimates.

7.3 Lessons learned

During this research project I, the author, have learned advanced and up-to-date specialized knowledge in the fields of Crypto implementation, Cryptanalysis, Coding Theory and Lattice-based Cryptography, as listed in Section 7.1 and evidenced by the papers in Part II.

In a more general sense, I have learned how to conduct research, and how to navigate my research areas' different methodologies. For example, I now know how to build up a theoretical framework (papers II and VI), how to apply it to a specific problem area and how to generate, aggregate and select results of relevance for publication. I have learned how to implement simulations (in C and in Rust) and disseminate the results. I have learned how the area of computer science in general likes to (or does not like to) structure its papers. This kind of knowledge is sometimes referred to as tribal knowledge, and I have learned much of the basics of what my tribe has offered to teach me, though I am still a beginner.

By my own independent discovery of the rejection sampling weakness of HQC and BIKE (paper V), I have demonstrated a capacity of "scholarly analysis and synthesis as well as to review and assess new and complex phenomena, issues and situations autonomously and critically" as one of my learning goals so succinctly puts it. The same can also be argued for my work with paper IV, though with less impressive results. In both cases there would not be a paper without my co-authors' experience, ability to see a way forward and willingness to put in the work to make it come through. Still, I believe I pulled my own weight in this regard. For all papers.

During this research project of 5+ years, I have trained my ability to formulate research questions, suggest new directions and conduct research according to a given and, many times, self-determined time-plan. This is not to say that I have achieved all my deadlines, but I have identified problems in advance and, though difficult to measure, I believe I have managed the risks associated in a responsible way.

Through not a trivial process of personal growth, I have learned to present talks and have oral communications, in English, about my research topic. I have presented at large conferences (CHES19, CRYPTO20, ISIT21) and been invited to present a longer version of my talk regarding paper III for a seminar. I have made successful collaborations with international colleagues at other universities, of which paper V is a particularly good and recent example.

In addition, I have learned to better identify my own shortcomings and take action to learn new knowledge in relation to those shortcomings. It is hard to find a particular example of this when almost all of this dissertation is a summary of knowledge I did not possess previous to starting this research project.

Learning to teach and to otherwise support the learning of others is something I have not had much exposure to in this project, being an industrial Ph.D. student. Regretting this, and in an attempt to compensate, I have offered and completed multiple seminars and shorter weakly presentations of various cryptography-related topics at the company which has so gracefully lent me to the university to do research. This has been very fun, and something I hope to continue with in the future.

Sometimes the discussion of research ethics and morale comes up. What is okay and what is not? I have considered responsible disclosure[2] in relation to papers III, V and VI but ultimately we decided that all those schemes are quite explicitly labeled experimental and if any implementations are being used in some product by some company they should be quite aware of the risks. Anyway, it does appear to be par for the course, in this tribe.

The whole idea of conducting research with the explicit interest of finding attacks and weaknesses in purely defensive and privacy-protecting computer code is, on the face of it, reprehensible. This of course is the complete opposite of the reality when one takes in the larger view of cryptanalytical- (or indeed security-) related research. This goes deeper than just "design or implementation issues can only be fixed if they are known"; To use a system, you place a level of trust in that system to keep your data safe. For cryptographic primitives and their implementations, the appropriate level of such trust can only ever be gained by relying on a quite substantial body of work, by well-respected researchers (and other would-be attackers, if they ever went public with their results). There is an analog to be made to medicinal research, where new drugs can sometimes take many years to get approval for general use, and only after all side-effects have been properly identified and investigated. But if any unexpected side-effect (or in our case, side-channel) appears, it could undermine the whole process, unless it is somehow handled responsibly. Similarly, if many well-respected researchers spend enough years trying to find attacks and weaknesses and only gain marginal results, the level of appropriate trust increases substantially. But, as always, "Absence of Evidence does not mean Evidence of Absence".

7.4 Looking forward

In this dissertation, we have seen (or will see in papers I to VI) how implementation issues, even on implementations and primitives that by now are a number of years old, and under a lot of scrutiny, can still harbor unexpected implementation issues.

Are there more to be found? Almost for certain, but it is my belief that the rate of pure timing attacks is decreasing since there has been a lot of successful effort into making implementations constant time. However, I predict that the area of cache-timing will still be able to uncover more vulnerabilities, just like what happened with classical crypto in the previous generation of asymmetric primitives.

Also, side-channels such as EM and power variations show no indication of slowing down either. Protecting against such attacks appears to be a much harder problem, as the whole threat model shifts in the attacker's favor.[3] This is why I

[2] Responsible disclosure is the act of providing the owners of a scheme or implementation with the details of an attack but also giving them enough time to fix the vulnerability before going public with the information.

[3] Masking schemes, which are a protection mechanism often used against such attacks, are often described by what order d they provide protection. For example, a first and second-order ($d = 1$ and $d = 2$, respectively) masking scheme split the computations into $d + 1$ shares, such that they

believe attacks will continue to be published for the foreseeable future.

Regarding BIKE and the static key vs. ephemeral key use cases, the authors of the scheme are quite clear in their statement that only the ephemeral key use case is supported. However, their IND-CCA security proof (conditional on a DFR analysis of the decoder) and their speed and effort towards fixing our rejection sampling vulnerability are telling of an ambition towards, perhaps, supporting also the static-key use-case, in the future. For this to happen, more work on theoretically determining the DFR bounds, has to be done. This is one area of future research that would be suitable for experts in coding theory, and perhaps more can be done in this area for interdisciplinary research teams.

Signature schemes are also a part of the NIST PQC effort, but none are part of this dissertation. This would also be an interesting direction to build upon my research, especially considering that NIST is launching a new call-for-proposals for PQC signature schemes.

Microarchitectural attacks, Microarchitectural Data Sampling attacks, Transient execution CPU vulnerabilities and other hardware security bugs in consumer-level processors have not been touched upon in this dissertation, though I believe they are, or could be, applicable to many of the implementations of the to-be-standardized KEM and signature schemes of the NIST PQC effort. This is an exciting research area and something I myself would like to have more experience in.

individually reveal nothing about the internal state of the computations. The masking order, d, can also be referred to as a security parameter but higher-order masking has adverse effects on the performance of a scheme, as well as being more difficult to implement. It appears that side-channels are often effective even in the presence of masking schemes, but with higher order, it makes the attacks more difficult, but never impossible.

References

[AD97] *A public-key cryptosystem with worst-case/ average-case equivalence.* http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.45.7341&rep=rep1&type=pdf. 1997, pp. 284–293.

[Agu+18] C. Aguilar-Melchor, O. Blazy, J. C. Deneuville, P. Gaborit, and G. Zemor. "Efficient Encryption from Random Quasi-Cyclic Codes". In: *IEEE Transactions on Information Theory* 64.5 (2018), pp. 3927–3943.

[Agu+22] C. Aguilar Melchor et al. *HQC.* Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions. National Institute of Standards and Technology, 2022.

[Ala+19] G. Alagic, J. Alperin-Sheriff, et al. *Status Report on the First Round of the NIST Post-Quantum Cryptography Standardization Process.* https://doi.org/10.6028/NIST.IR.8240. 2019.

[Ala+22] G. Alagic, D. Apon, et al. *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process.* https://doi.org/10.6028/NIST.IR.8413-upd1. 2022.

[Alb+22] M. R. Albrecht et al. *Classic McEliece.* Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions. National Institute of Standards and Technology, 2022.

[AM87] C. M. Adams and H. Meijer. "Security-related comments regarding McEliece's public-key cryptosystem". In: *Conference on the Theory and Application of Cryptographic Techniques.* Springer, 1987, pp. 224–228.

[Ara+22] N. Aragon et al. *BIKE*. Tech. rep. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions`. National Institute of Standards and Technology, 2022.

[Bar+12] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache. "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures". In: *Proceedings of the IEEE* 100.11 (2012), pp. 3056–3076.

[BB03] D. Brumley and D. Boneh. "Remote Timing Attacks Are Practical". In: *USENIX Security 2003: 12th USENIX Security Symposium*. Washington, DC, USA: USENIX Association, Aug. 2003.

[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.

[Bel+97] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. "A concrete security treatment of symmetric encryption". In: *Proceedings 38th Annual Symposium on Foundations of Computer Science*. IEEE. 1997, pp. 394–403.

[Ber+17a] D. J. Bernstein, C. Chuengsatiansup, T. Lange, and C. van Vredendaal. *NTRU Prime*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`. National Institute of Standards and Technology, 2017.

[Ber+17b] D. J. Bernstein, J. Fried, N. Heninger, P. Lou, and L. Valenta. *Post-quantum RSA-Encryption*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`. National Institute of Standards and Technology, 2017.

[Ber+18] D. J. Bernstein, L. G. Bruinderink, T. Lange, and L. Panny. "HILA5 Pindakaas: On the CCA Security of Lattice-Based Encryption with Error Correction". In: *AFRICACRYPT 18: 10th International Conference on Cryptology in Africa*. Ed. by A. Joux, A. Nitaj, and T. Rachidi. Vol. 10831. Lecture Notes in Computer Science. Marrakesh, Morocco: Springer, Heidelberg, Germany, May 2018, pp. 203–216.

[Ber10] D. J. Bernstein. "Grover vs. Mceliece". In: *Proceedings of the Third International Conference on Post-Quantum Cryptography*. PQCrypto'10. Darmstadt, Germany: Springer-Verlag, 2010, pp. 73–80.

[Bha+21] S. Bhasin, J.-P. D'Anvers, D. Heinz, T. Pöppelmann, and M. Van Beirendonck. "Attacking and defending masked polynomial comparison for lattice-based cryptography". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), pp. 334–359.

[Ble98] D. Bleichenbacher. "Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS# 1". In: *Annual International Cryptology Conference*. Springer. 1998, pp. 1–12.

[Bol+14] A. Boldyreva, J. P. Degabriele, K. G. Paterson, and M. Stam. "On Symmetric Encryption with Distinguishable Decryption Failures". In: *Fast Software Encryption – FSE 2013*. Ed. by S. Moriai. Vol. 8424. Lecture Notes in Computer Science. Singapore: Springer, Heidelberg, Germany, Mar. 2014, pp. 367–390.

[Bru+16] L. G. Bruinderink, A. Hülsing, T. Lange, and Y. Yarom. "Flush, Gauss, and Reload - A Cache Attack on the BLISS Lattice-Based Signature Scheme". In: *Cryptographic Hardware and Embedded Systems – CHES 2016*. Ed. by B. Gierlichs and A. Y. Poschmann. Vol. 9813. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2016, pp. 323–345.

[BT11] B. B. Brumley and N. Tuveri. "Remote Timing Attacks Are Still Practical". In: *Computer Security - ESORICS 2011 - 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings*. Ed. by V. Atluri and C. Díaz. Vol. 6879. Lecture Notes in Computer Science. Springer, 2011, pp. 355–371.

[CD23] W. Castryck and T. Decru. "An efficient key recovery attack on SIDH". In: Springer-Verlag, 2023.

[Che+20] C. Chen et al. *NTRU*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`. National Institute of Standards and Technology, 2020.

[CRR03] S. Chari, J. R. Rao, and P. Rohatgi. "Template attacks". 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. 13–28.

[DAn+19] J.-P. D'Anvers, M. Tiepelt, F. Vercauteren, and I. Verbauwhede. "Timing attacks on error correcting codes in post-quantum schemes". In: *Proceedings of ACM Workshop on Theory of Implementation Security Workshop*. 2019, pp. 2–9.

[Den03] A. W. Dent. “A designer’s guide to KEMs”. In: *IMA International Conference on Cryptography and Coding*. Springer. 2003, pp. 133–151.

[DGK23] N. Drucker, S. Gueron, and D. Kostic. “To reject or not reject-that is the question. The case of BIKE post quantum KEM”. In: (2023).

[DH76] W. Diffie and M. Hellman. “New directions in cryptography”. In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654.

[Din+16] J. Ding, S. Alsayigh, S. RV, S. Fluhrer, and X. Lin. *Leakage of Signal function with reused keys in RLWE key exchange*. Cryptology ePrint Archive, Report 2016/1176. `http://eprint.iacr.org/2016/1176`. 2016.

[Duc+13] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky. “Lattice signatures and bimodal Gaussians”. In: *Annual Cryptology Conference*. Springer. 2013, pp. 40–56.

[DVV18] J.-P. D’Anvers, F. Vercauteren, and I. Verbauwhede. *On the impact of decryption failures on the security of LWE/LWR based schemes*. Cryptology ePrint Archive, Report 2018/1089. `https://eprint.iacr.org/2018/1089`. 2018.

[Ell70] J. H. Ellis. “The possibility of secure non-secret digital encryption”. In: *UK Communications Electronics Security Group* 8 (1970).

[Flu16] S. Fluhrer. *Cryptanalysis of ring-LWE based key exchange with key share reuse*. Cryptology ePrint Archive, Report 2016/085. `https://eprint.iacr.org/2016/085`. 2016.

[FO13] E. Fujisaki and T. Okamoto. “Secure integration of asymmetric and symmetric encryption schemes”. In: *Journal of cryptology* 26.1 (2013), pp. 80–101.

[FO99] E. Fujisaki and T. Okamoto. “Secure Integration of Asymmetric and Symmetric Encryption Schemes”. In: *Advances in Cryptology – CRYPTO’99*. Ed. by M. J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1999, pp. 537–554.

[Gal62] R. Gallager. “Low-density parity-check codes”. In: *IRE Transactions on information theory* 8.1 (1962), pp. 21–28.

[GGH97] O. Goldreich, S. Goldwasser, and S. Halevi. “Eliminating decryption errors in the Ajtai-Dwork cryptosystem”. In: *Annual International Cryptology Conference*. Springer, 1997, pp. 105–111.

[GJJ22] Q. Guo, A. Johansson, and T. Johansson. “A Key-Recovery Side-Channel Attack on Classic McEliece Implementations”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022.4 (2022), pp. 800–827.

[GJN19] Q. Guo, T. Johansson, and A. Nilsson. *A Generic Attack on Lattice-based Schemes using Decryption Errors with Application to ss-ntru-pke*. Cryptology ePrint Archive, Report 2019/043. `https://eprint.iacr.org/2019/043`. 2019.

[GJS16] Q. Guo, T. Johansson, and P. Stankovski. “A Key Recovery Attack on MDPC with CCA Security Using Decoding Errors”. In: *Advances in Cryptology – ASIACRYPT 2016, Part I*. Ed. by J. H. Cheon and T. Takagi. Vol. 10031. Lecture Notes in Computer Science. Hanoi, Vietnam: Springer, Heidelberg, Germany, Dec. 2016, pp. 789–815.

[GJW18] Q. Guo, T. Johansson, and P. S. Wagner. “A Key Recovery Reaction Attack on QC-MDPC”. In: *IEEE Trans. on Inf. Theory* (2018).

[GLG22] G. Goy, A. Loiseau, and P. Gaborit. “A New Key Recovery Side-Channel Attack on HQC with Chosen Ciphertext”. In: *Post-Quantum Cryptography*. Ed. by J. H. Cheon and T. Johansson. Cham: Springer International Publishing, 2022, pp. 353–371.

[GM19] S. Goldwasser and S. Micali. “Probabilistic encryption & how to play mental poker keeping secret all partial information”. In: *Providing sound foundations for cryptography: on the work of Shafi Goldwasser and Silvio Micali*. 2019, pp. 173–201.

[GN07] N. Gama and P. Q. Nguyen. “New Chosen-Ciphertext Attacks on NTRU”. In: *PKC 2007: 10th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by T. Okamoto and X. Wang. Vol. 4450. Lecture Notes in Computer Science. Beijing, China: Springer, Heidelberg, Germany, Apr. 2007, pp. 89–106.

[Gro96] L. K. Grover. “A Fast Quantum Mechanical Algorithm for Database Search”. In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. STOC ’96. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, pp. 212–219.

[Gro97] L. K. Grover. “Quantum Mechanics Helps in Searching for a Needle in a Haystack”. In: *Phys. Rev. Lett.* 79 (2 July 1997), pp. 325–328.

[GST14] D. Genkin, A. Shamir, and E. Tromer. “RSA key extraction via low-bandwidth acoustic cryptanalysis”. In: *Annual cryptology conference*. Springer. 2014, pp. 444–461.

[GST17] D. Genkin, A. Shamir, and E. Tromer. “Acoustic cryptanalysis”. In: *Journal of Cryptology* 30.2 (2017), pp. 392–443.

[Ham+21] M. Hamburg et al. “Chosen Ciphertext k-Trace Attacks on Masked CCA2 Secure Kyber”. In: *IACR TCHES* 2021.4 (2021). `https://tches.iacr.org/index.php/TCHES/article/view/9061`, pp. 88–113.

[HGS99] C. Hall, I. Goldberg, and B. Schneier. “Reaction Attacks against several Public-Key Cryptosystems”. In: *ICICS 99: 2nd International Conference on Information and Communication Security*. Ed. by V. Varadharajan and Y. Mu. Vol. 1726. Lecture Notes in Computer Science. Sydney, Australia: Springer, Heidelberg, Germany, Nov. 1999, pp. 2–12.

[HHK17] D. Hofheinz, K. Hövelmanns, and E. Kiltz. “A Modular Analysis of the Fujisaki-Okamoto Transformation”. In: *TCC 2017: 15th Theory of Cryptography Conference, Part I*. Ed. by Y. Kalai and L. Reyzin. Vol. 10677. Lecture Notes in Computer Science. Baltimore, MD, USA: Springer, Heidelberg, Germany, Nov. 2017, pp. 341–371.

[HHM22] K. Hövelmanns, A. Hülsing, and C. Majenz. *Failing gracefully: Decryption failures and the Fujisaki-Okamoto transform*. 2022.

[HLS21] C. Hlauschek, N. Lahr, and R. L. Schröder. *On the Timing Leakage of the Deterministic Re-encryption in HQC KEM*. Cryptology ePrint Archive, Report 2021/1485, version 20211115:124514 (posted 1636980314 15-Nov-2021 12:45:14 UTC). `https://eprint.iacr.org/2021/1485/20211115:124514`. Aug. 2021.

[Hon+02] J. Hong, J. W. Han, D. Kwon, and D. Han. *Chosen-Ciphertext Attacks on Optimized NTRU*. 2002.

[How+03] N. Howgrave-Graham et al. “The Impact of Decryption Failures on the Security of NTRU Encryption”. In: *Advances in Cryptology – CRYPTO 2003*. Ed. by D. Boneh. Vol. 2729. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2003, pp. 226–246.

[HPS96] J. Hoffstein, J. Pipher, and J. H. Silverman. "NTRU: a new high speed public key cryptosystem". In: *preliminary draft presented at the rump session of Crypto 96* (1996). https://www.ntru.org/f/hps96.pdf.

[HS00] J. Hoffstein and J. H. Silverman. "Protecting NTRU Against Chosen Ciphertext and Reaction Attacks". In: 2000.

[HS99] J. Hoffstein and J. H. Silverman. *Reaction attacks against the NTRU public key cryptosystem*. Tech. rep. Technical Report 15, NTRU Cryptosystems, 1999.

[JJ00] É. Jaulmes and A. Joux. "A Chosen-Ciphertext Attack against NTRU". In: *Advances in Cryptology — CRYPTO 2000*. Ed. by M. Bellare. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 20–35.

[Kau+16] T. Kaufmann, H. Pelletier, S. Vaudenay, and K. Villegas. "When Constant-Time Source Yields Variable-Time Binary: Exploiting Curve25519-donna Built with MSVC 2015". In: *Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings*. Ed. by S. Foresti and G. Persiano. Vol. 10052. Lecture Notes in Computer Science. 2016, pp. 573–582.

[Kel98] T. Kelly. "THE MYTH OF THE SKYTALE". In: *Cryptologia* 22.3 (1998), pp. 244–260. eprint: https://doi.org/10.1080/0161-119891886902.

[KJJ99] P. Kocher, J. Jaffe, and B. Jun. "Differential Power Analysis". In: *CRYPTO 1999*. Boston, MA: Springer US, 1999, pp. 388–397.

[Koc96] P. C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In: *Advances in Cryptology – CRYPTO'96*. Ed. by N. Koblitz. Vol. 1109. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1996, pp. 104–113.

[KS05] F. Koeune and F.-X. Standaert. "A Tutorial on Physical Security and Side-Channel Attacks". In: *Foundations of Security Analysis and Design III: FOSAD 2004/2005 Tutorial Lectures*. Ed. by A. Aldini, R. Gorrieri, and F. Martinelli. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 78–108.

[LHT16] A. Langley, M. Hamburg, and S. Turner. "RFC 7748: Elliptic curves for security". In: *Internet Research Task Force (IRTF)* (2016).

[McE78] R. J. McEliece. "A Public-Key Cryptosystem Based On Algebraic Coding Theory". In: *DSN Progress Report 42–44* (1978), pp. 114–116.

[Mos18] M. Mosca. “Cybersecurity in an Era with Quantum Computers: Will We Be Ready?” In: *IEEE Security & Privacy* 16.5 (2018), pp. 38–41.

[NDJ21] K. Ngo, E. Dubrova, and T. Johansson. “Breaking masked and shuffled CCA secure Saber KEM by power analysis”. In: *Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security*. 2021, pp. 51–61.

[Ngo+21] K. Ngo, E. Dubrova, Q. Guo, and T. Johansson. “A Side-Channel Attack on a Masked IND-CCA Secure Saber KEM Implementation”. In: *IACR TCHES* 2021.4 (2021). `https://tches.iacr.org/index.php/TCHES/article/view/9079`, pp. 676–707.

[Nie86] H. Niederreiter. “Knapsack-type cryptosystems and algebraic coding theory”. In: *Prob. Contr. Inform. Theory* 15.2 (1986), pp. 157–166.

[NIS16] NIST. *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*. `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf`. 2016.

[NIS18] NIST. *Post-Quantum Cryptography FAQs*. 2018. URL: `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/faqs` (visited on 04/10/2023).

[NIS22] NIST. *Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process*. `https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf`. 2022.

[Por17] T. Pornin. *Why Constant-Time Crypto?* 2017. URL: `https://www.bearssl.org/constanttime.html` (visited on 04/11/2023).

[PT19] T. B. Paiva and R. Terada. “A Timing Attack on the HQC Encryption Scheme”. In: *SAC 2019*. Ed. by K. G. Paterson and D. Stebila. Vol. 11959. LNCS. Springer, Heidelberg, Aug. 2019, pp. 551–573.

[Raj+22] G. Rajendran, P. Ravi, J.-P. D’Anvers, S. Bhasin, and A. Chattopadhyay. *Pushing the Limits of Generic Side-Channel Attacks on LWE-based KEMs - Parallel PC Oracle Attacks on Kyber KEM and Beyond*. Cryptology ePrint Archive, Paper 2022/931. 2022.

[Rav+20] P. Ravi, S. S. Roy, A. Chattopadhyay, and S. Bhasin. “Generic Side-channel attacks on CCA-secure lattice-based PKE and KEMs”. In: *IACR TCHES* 2020.3 (2020). `https://tches.iacr.org/index.php/TCHES/article/view/8592`, pp. 307–335.

[Rav+21] P. Ravi, S. Bhasin, S. S. Roy, and A. Chattopadhyay. “On Exploiting Message Leakage in (Few) NIST PQC Candidates for Practical Message Recovery Attacks”. In: *IEEE Transactions on Information Forensics and Security* 17 (2021), pp. 684–699.

[Rav+22] P. Ravi, A. Chattopadhyay, J. P. D’Anvers, and A. Baksi. *Side-channel and Fault-injection attacks over Lattice-based Post-quantum Schemes (Kyber, Dilithium): Survey and New Results*. Cryptology ePrint Archive, Paper 2022/737. `https://eprint.iacr.org/2022/737`. 2022.

[Reg05] O. Regev. “On lattices, learning with errors, random linear codes, and cryptography”. In: *37th Annual ACM Symposium on Theory of Computing*. Ed. by H. N. Gabow and R. Fagin. Baltimore, MA, USA: ACM Press, May 2005, pp. 84–93.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”. In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126.

[RV14] M. Rydelnik and M. Vanlaningham. *The Moody Bible Commentary*. Ed. by M. Rydelnik and M. Vanlaningham. Moody Publishers, 2014.

[Sch+17] J. M. Schanck, A. Hulsing, J. Rijneveld, and P. Schwabe. *NTRU-HRSS-KEM*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`. National Institute of Standards and Technology, 2017.

[Sch+22a] T. Schamberger, L. Holzbaur, J. Renner, A. Wachter-Zeh, and G. Sigl. “A Power Side-Channel Attack on the Reed-Muller Reed-Solomon Version of the HQC Cryptosystem”. In: *Post-Quantum Cryptography*. Ed. by J. H. Cheon and T. Johansson. Cham: Springer International Publishing, 2022, pp. 327–352.

[Sch+22b] P. Schwabe et al. *CRYSTALS-KYBER*. Tech. rep. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`. National Institute of Standards and Technology, 2022.

[Sch18] J. M. Schanck. “A comparison of NTRU variants”. In: *Cryptology ePrint Archive* (2018).

[Sen21] N. Sendrier. “Secure sampling of constant-weight words–application to bike”. In: *Cryptology ePrint Archive* (2021).

[Sha49] C. E. Shannon. “Communication theory of secrecy systems”. In: *The Bell System Technical Journal* 28.4 (1949), pp. 656–715.

[Sho01] V. Shoup. *A Proposal for an ISO Standard for Public Key Encryption*. Cryptology ePrint Archive, Paper 2001/112. `https://eprint.iacr.org/2001/112`. 2001.

[Sho94] P. Shor. “Algorithms for quantum computation: discrete logarithms and factoring”. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994, pp. 124–134.

[Sin03] S. Singh. *The Code Book: The Secrets Behind Codebreaking*. Random House Children’s Books, 2003.

[SM15] T. Schneider and A. Moradi. “Leakage assessment methodology: A clear roadmap for side-channel evaluations”. In: *Cryptographic Hardware and Embedded Systems–CHES 2015: 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings 17*. Springer. 2015, pp. 495–513.

[Sma16] N. P. Smart. *Cryptography Made Simple*. Information Security and Cryptography. Springer, Heidelberg, Germany, 2016.

[Str10] F. Strenzke. “A timing attack against the secret permutation in the McEliece PKC”. In: *International Workshop on Post-Quantum Cryptography*. Springer. 2010, pp. 95–107.

[Str13] F. Strenzke. “Timing attacks against the syndrome inversion in code-based cryptosystems”. In: *International Workshop on Post-Quantum Cryptography*. Springer. 2013, pp. 217–230.

[Tan+22] Y. Tanaka, R. Ueno, K. Xagawa, A. Ito, J. Takahashi, and N. Homma. *Multiple-Valued Plaintext-Checking Side-Channel Attacks on Post-Quantum KEMs*. Cryptology ePrint Archive, Paper 2022/940. 2022.

[Uen+22] R. Ueno, K. Xagawa, Y. Tanaka, A. Ito, J. Takahashi, and N. Homma. “Curse of Re-encryption: A Generic Power/EM Analysis on Post-Quantum KEMs”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022.1 (2022), pp. 296–322.

[Waf+19] G. Wafo-Tapa, S. Bettaieb, L. Bidoux, P. Gaborit, and E. Marcatel. *A Practicable Timing Attack Against HQC and its Countermeasure*. Cryptology ePrint Archive, Report 2019/909. `https://eprint.iacr.org/2019/909`. 2019.

[WND22a] R. Wang, K. Ngo, and E. Dubrova. "Making Biased DL Models Work: Message and Key Recovery Attacks on Saber Using Amplitude-Modulated EM Emanations". In: *Cryptology ePrint Archive* (2022).

[WND22b] R. Wang, K. Ngo, and E. Dubrova. "Side-channel analysis of Saber KEM using amplitude-modulated EM emanations". In: *Cryptology ePrint Archive* (2022).

[Xu+22] Z. Xu, O. Pemberton, S. S. Roy, D. Oswald, W. Yao, and Z. Zheng. "Magnifying Side-Channel Leakage of Lattice-Based Cryptosystems With Chosen Ciphertexts: The Case Study of Kyber". In: *IEEE Transactions on Computers* 71.9 (2022), pp. 2163–2176.

[Zha+17] Z. Zhang, C. Chen, J. Hoffstein, and W. Whyte. *NTRUEncrypt*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`. National Institute of Standards and Technology, 2017.

[Zha+19] Z. Zhang, C. Chen, J. Hoffstein, W. Whyte, et al. *NTRUEncrypt*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`. National Institute of Standards and Technology, 2019.

Included Publications

Error Amplification in Code-based Cryptography

Abstract

Code-based cryptography is one of the main techniques enabling cryptographic primitives in a post-quantum scenario. In particular, the MDPC scheme is a basic scheme from which many other schemes have been derived. These schemes rely on iterative decoding in the decryption process and thus have a certain small probability p of having a decryption (decoding) error.

In this paper we show a very fundamental and important property of code-based encryption schemes. Given one initial error pattern that fails to decode, the time needed to generate another message that fails to decode is strictly *much* less than $1/p$. We show this by developing a method for fast generation of undecodable error patterns (error pattern chaining), which additionally proves that a measure of closeness in ciphertext space can be exploited through its strong linkage to the difficulty of decoding these messages. Furthermore, if side-channel information is also available (time to decode), then the initial error pattern no longer needs to be given since one can be easily generated in this case.

These observations are fundamentally important because they show that a, say, 128-bit encryption scheme is not inherently safe from reaction attacks even if it employs a decoder with a failure rate of 2^{-128}. In fact, unless explicit protective measures are taken, having a failure rate at all – of any magnitude – can pose a security problem because of the error amplification effect of our method.

A key-recovery reaction attack was recently shown on the MDPC scheme as well as similar schemes, taking advantage of decoding errors in order to recover the secret key. It was also shown that knowing the number of iterations in the iterative decoding step, which could be received in a timing attack, would also enable and

A. Nilsson, T. Johansson, and P. Stankovski Wagner. "Error Amplification in Code-based Cryptography". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019.1 (Nov. 2018), pp. 238–258 enhance such an attack. In this paper we apply our error pattern chaining method to show how to improve the performance of such reaction attacks in the CPA case. We show that after identifying a single decoding error (or a decoding step taking more time than expected in a timing attack), we can adaptively create new error patterns that have a much higher decoding error probability than for a random error. This leads to a significant improvement of the attack based on decoding errors in the CPA case and it also gives the strongest known attack on MDPC-like schemes, both with and without using side-channel information.

1 Introduction

Future quantum computers will be able to break cryptography based on integer factorization and discrete log in polynomial time. This fact pushed cryptographic research to focus on post-quantum solutions, i.e., finding new primitives based on more well suited mathematical problems that may still be difficult to solve for a quantum computer. That is, the new primitives must be efficiently computable on classical computers, but an adversary must not be able to break the scheme even if she has access to a powerful quantum computer. This area of research is called post-quantum cryptography [Ber09].

Code-based cryptography leverages difficult problems in coding theory and is one of the main techniques enabling cryptographic primitives in this post-quantum scenario. The classic idea in code-based cryptography is the McEliece scheme from 1978 [McE78]. However, recently much more attractive schemes have been proposed, centered around the quasi-cyclic medium density parity check (QC-MDPC) scheme [Mis+13]. It has a significantly smaller public key than the McEliece scheme. The QC-MDPC scheme is, like many other similar variants of code-based cryptographic encryption schemes, based on iterative decoding; it has a certain probability of decryption failure (for proposed parameters in the range of 10^{-4} to 10^{-8} [HMG13], depending on decoding algorithm, implementation and chosen parameters). Also, as the decryption step is iterative, it will require a varying number of rounds before finishing, which leads to varying decoding time in a standard implementation. This fact opens the door for possible timing attacks based on the number of required decoding rounds. It was shown by Guo, Johansson and Stankovski [GJS16], that decoding errors can be used to reconstruct the secret key. The attack breaks the chosen-ciphertext attack (CCA) security of the scheme and provides attackers with a key recovery attack that requires submitting 200-350 million ciphertexts for decryption. This was based on proposed parameters for 80-bit security and the decryption device using an iterative decoding algorithm with a decoding error probability around 10^{-4}. Better decoding algorithms (with higher complexity) could lower the decoding error and this would lead to an increase in the complexity of the attack.

Let us briefly recall the idea behind the Guo et al. attack. The authors identified a dependency between the secret key and decoding failures. They found that if there were two ones in the key at (cyclic) distance d and the error pattern also contained two ones at a distance d, then the probability for decoding error is smaller than in the opposite case. This observation was used to build a so called distance spectrum of the secret key. A distance spectrum can be viewed as the set of all distances between any two non-zero bit positions in the key. In order to build this distance spectrum, the authors simply aggregated the distance spectrum of each bit pattern in messages which led to a decoding failure. This distance spectrum can, in a reconstruction step, be used to directly determine the secret key.

In the recent NIST post-quantum standardization project [Che+16], a number of code-based schemes have been submitted. Looking through these submissions, one can see that the above described attack has impact on the security analysis of such schemes. The attack has also been generalized, analyzed and improved in different directions. In [Fab+17] the attack was extended to break the QC-LDPC McEliece scheme presented in [BBC08]. A similar attack on LEDApkc appears in [FHZ18].

In the Guo et al. paper, it was pointed out that the attack would also extend to a timing attack. This was fully examined by Eaton et al. in [Eat+18]. Not only did they provide the framework for a timing attack, but they also gave an extended theoretical treatment of the attack and showed the dependence on the syndrome weight in decoding.

1.1 Contributions

In this paper we show a method for generating large quantities of error patterns that fail to decode for any given iterative decoder. Given one initial error pattern that fails to decode, the time needed to generate another message that fails to decode is almost negligible. Our method for fast generation of undecodable error patterns (error pattern chaining) additionally proves that there is a measure of closeness in ciphertext space such that similar ciphertext messages are roughly equally difficult to decode. In addition, when side-channel information such as decoding time or number of iteration used during decoding is also available, then the initial error pattern no longer needs to be given since one can easily be generated instead.

These observations are fundamentally important because they show that a, say, 128-bit encryption scheme is not inherently safe from reaction attacks even if it employs a decoder with a failure rate of 2^{-128}. Extrapolating from the general ideas of this paper and those of Guo et al., it would seem that failure rates, regardless of their magnitude, appear to convey a potential for security problems. We further argue that using very low failure rates is, on its own, not enough to discount such weaknesses, in any cryptographic system.

We also apply our findings towards improving the attacks of Guo et al. and Eaton et al. We use error pattern chaining to explore the possibility of artificially and adaptively increasing the error probability and improve on previous works by extracting much more information from *all* decoding attempts. We do this in a chosen plaintext attack (CPA) setting. The attacks are improved in two ways: 1) it increases the possibility of finding another non-decodable error pattern and 2) it enables us to extract more information from patterns that *can* be decoded. If we additionally consider a timing attack we can improve the attack even further, both in finding initial error patterns and in performing the main attack.

Simulations show that a distance spectrum can be built from this chain of errors in a similar manner as in the original attack. This method enables us to use more than an order of magnitude fewer decoding trials to recover the secret key in the CPA setting, compared to the original attack of Guo et al. in [GJS16].

Comparing our work with the recent work of Eaton et al. [Eat+18], we improve upon their original attack even *without* using private information such as the syndrome weight. When Eaton et al. convert their attack to a timing-based side-channel attack, they require $2^{25} \approx 33.5$M ciphertexts to fully recover the key for standard parameters of 80-bit security. Using our idea of chains of related error patterns, our new attack requires less than 12M ciphertexts without using any side-channel information, and less than 8M ciphertexts with side-channel information.

Taking into consideration the fact that [GJS] has shown that a fully correct distance spectrum is not necessary to recover the secret key, we show that as few as 310 000 ciphertexts are necessary to perform a successful attack, using a distance spectrum with 900 errors, for 80-bit security.

It is also clear that this new adaptive approach will be even more beneficial when the decoding error is very small, which could be expected in a scheme proposed for actual use.

1.2 Paper Organization

In Section 2 we give some background of code-based cryptography, QC-MDPC and the original reaction attack by Guo et al. [GJS16]. In Section 3 we provide the theory behind our new method of performing error rate amplification. We also present how to use this method in an attack against QC-MDPC. Then, in Section 4, we describe the results we obtained by implementing the new method and testing it against a few different decoder implementations. Finally, we conclude the paper in Section 5.

2 Background

In this section we briefly give some of the basic background information necessary to follow this paper.

2.1 Coding Theory and Public-Key Cryptography

We review some basics from coding theory and show its application to public-key cryptography.

Definition 1 (Linear codes). *An $[n, k]$ linear code $\mathcal{C}$ over a finite field $\mathbb{F}_q$ is a linear subspace of $\mathbb{F}_q^n$ of dimension k.*

Definition 2 (Generator matrix). *A $k \times n$ matrix G with entries from $\mathbb{F}_q$ having rowspan $\mathcal{C}$ is a generator matrix for the $[n, k]$ linear code $\mathcal{C}$.*

The code $\mathcal{C}$ is the kernel of an $(n - k) \times n$ matrix H called a *parity-check matrix* of $\mathcal{C}$. We have $\mathbf{c}H^{\mathrm{T}} = \mathbf{0}$, if and only if $\mathbf{c} \in \mathcal{C}$, where H^{T} is the transpose of H.

The code $\mathcal{C}$ can be represented by different generator matrices. An important one is the systematic form, i.e., when each input symbol is directly represented in a position in the codeword. One can find a $k \times k$ submatrix of G forming the identity matrix and after a permutation one can consider G of the form $G = \begin{pmatrix} I & P \end{pmatrix}$. If G has such a systematic form then $H = \begin{pmatrix} -P^{\mathrm{T}} & I \end{pmatrix}$.

We now only consider binary codes, i.e. $q = 2$. The Hamming weight $w_H(\mathbf{x})$ of a binary vector in $\mathbf{x} \in \mathbb{F}_2^n$ is the number of nonzero entries in the vector. The minimum (Hamming) distance of the code $\mathcal{C}$ is defined as $d = \min_{\mathbf{x},\mathbf{y} \in \mathcal{C}} w_H(\mathbf{x} - \mathbf{y})$, where $\mathbf{x} \neq \mathbf{y}$.

Definition 3 (Quasi-cyclic codes). *An $[n, k]$ quasi-cylic (QC) code $\mathcal{C}$ is a linear code such that for some integer n_0, every cyclic shift of any codeword by n_0 steps is again a codeword.*

In particular, if $n = n_0 k$ for a QC code, then a generator matrix of the form

$$G = \begin{pmatrix} I & P_0 & P_1 & \cdots & P_{n_0 - 1} \end{pmatrix}$$

is a possible representation of a QC code, where P_i is a $k \times k$ cyclic matrix, i.e. the rows (or columns) of P are obtained by cyclic rotations of the first row. Also, the algebra of $k \times k$ binary circulant matrices is isomorphic to the algebra of polynomials modulo $x^k + 1$ over $\mathbb{F}_2$, allowing an alternative description.

Another useful class of codes is the class of Low-Density Parity-Check codes (LDPC codes), defined as linear codes that admit a sparse parity-check matrix H, where sparsity means that each row of H has at most w ones, for some small w. This sparse matrix can be represented in the form of a bipartite graph, that consists of $n - k$ upper nodes (named "check nodes") representing the $n - k$ parity equations and n lower nodes (named "variable nodes") representing the n codeword positions. A variable node is connected to a check node if the variable is present in that parity check. Each check node is then connected to w variable nodes. We call this graph representation a "Tanner" graph, which is a frequently used term in work on iterative decoding algorithms.

2.2 McEliece Cryptosystem

In 1978 McEliece showed how a public key cryptosystem (PKC) could be constructed using tools from coding theory. We briefly describe the original McEliece PKC here. This scheme uses three matrices G, S, P, where G is a $k \times n$ generator matrix of a binary $[n, k, 2t + 1]$ linear code. The original and still secure proposal in [McE78] is to use Goppa codes (see [Gop70; MS77]). Then S is a $k \times k$ random binary non-singular matrix (called the scrambling matrix), and P is an $n \times n$ random permutation matrix (called the permutation matrix). As designers we compute the new $k \times n$ matrix $G' = SGP$. The scheme works as follows. The private key is (G, S, P) and the public key is (G', t). In encryption, a message $\mathbf{m}$ is mapped to a ciphertext $\mathbf{c}$ by $\mathbf{c} = \mathbf{m}G' + \mathbf{e}$, where $\mathbf{c}$ is the n-bit ciphertext, $\mathbf{m}$ is the k-bit plaintext and $\mathbf{e}$ an n-bit error vector with (Hamming) weight t. In decryption, one uses an efficient decoding algorithm for Goppa codes to decode $\mathbf{c}$ to find the error $\mathbf{e}P^{-1}$, and recover $\mathbf{m}S$ and thus $\mathbf{m}$.

Knowing the description of the selected Goppa code G allows efficient decoding, as there are many efficient decoding algorithms for this problem running in polynomial time. But knowing only the public key G', the attacker is facing a decoding problem for a code that looks like a random code, which is a difficult problem. The attacker can either try to decode an intercepted ciphertext (message recovery attack) or try to recover the secret matrix G from the public matrix G' (key recovery attack).

2.3 The QC-MDPC Public Key Encryption System

In [Mis+13], a new powerful version of the McEliece PKC was proposed. It has a simpler description as it does not use permutation and scrambling matrices as in the original McEliece construction or in other proposed generalizations [Bal+07; LJ12]. The idea is to use codes that allow iterative decoding. In coding theory, this typically involves LDPC codes, but for a cryptographic scheme this is not secure. LDPC codes have parity-checks with very small Hamming weight and such parity-checks in a given LDPC code correspond to codewords in the dual code. Since the dual code can be computed, it is also easy to find low-weight codewords in the dual code and thus the low-weight parity checks. The solution proposed in [Mis+13] is to increase the weight of the parity checks to a larger value, but still small in comparison with the dimension of the code. This makes the task of finding low-weight codewords in the dual code much more costly. In this way, key-recovery attacks by searching for low weight codewords are avoided.

Such codes with increased parity-check weight are called *Moderate-Density Parity-Check* codes (MDPC codes), and they can be decoded with the same decoding algorithms used to decode LDPC codes. The quasi-cyclic variant of MDPC codes are called QC-MDPC codes. These are of special interest, since the quasi-cyclic property allows us to represent the code by a single row of the generator matrix. Since the public key is a generator matrix, this gives us very compact keys. We will now describe the different steps of the QC-MDPC public key cryptosystem as proposed in [Mis+13]. We will restrict ourselves to $n_0 = 2$, corresponding to parameters $r = k = n/2$.

Key Generation

1. Choose an $[n, n/2]$ code in the QC-MDPC family, described by the parity-check matrix $H \in \mathbb{F}_2^{r \times n}$, such that
$$H = \begin{pmatrix} H_0 & H_1 \end{pmatrix},$$
where each H_i is a circulant $r \times r$ matrix with weight w_i in each row and with $w = \sum w_i$.

2. Generate the public key $G \in \mathbb{F}_2^{(n-r) \times n}$ from H as,
$$G = \begin{pmatrix} I & P \end{pmatrix},$$
where
$$P = \left(\left(H_1^{-1} H_0\right)^{\mathrm{T}} \right).$$

Recall, the QC-MDPC scheme has no need for permutation or scrambling matrices.

Encryption

Let $\mathbf{m} \in \mathbb{F}_2^{(n-r)}$ be the plaintext. Multiply $\mathbf{m}$ with the public key G and add noise within the correction radius t of the code, i.e., $\mathbf{c} = \mathbf{m}G + \mathbf{e}$, where $w_H(\mathbf{e}) \leq t$. The parameter t is obtained from the error correcting capability of the decoding algorithm for the MDPC code [Mis+13]. The error vector is uniformly chosen among all binary n-tuples with $w_H(\mathbf{e}) \leq t$.

2.4 Decryption

Let $\mathbf{c} \in \mathbb{F}_2^n$ be a received ciphertext. Given the secret low-weight parity check matrix H, a low-complexity decoding procedure is used to obtain the plaintext $\mathbf{m}$.

The authors of [Mis+13] propose the use of Gallager's bit-flipping algorithm [Gal62] for the decoding of MDPC codes. This bit-flipping procedure is vital to the proposed key recovery attack under consideration. The decoding step works as follows:

1. Compute the syndrome $\mathbf{s} = \mathbf{c}H^T$. Since $\mathbf{m}H^T = \mathbf{0}$, this is equivalently expressed as $\mathbf{s} = \mathbf{e}H^T$. Consider the Tanner graph corresponding to H. Create a counter with an initial value 0 for each variable node.

2. Run through all parity-check equations (rows of H and/or check nodes in the graph) and for every variable node connected to an unsatisfied check node, increase its corresponding counter by one.

3. Run through all variable nodes and flip its value if its counter is larger than a predetermined threshold δ_j.

4. If all the equations are satisfied, or iteration counter j reached a maximum value, then stop; otherwise, set all counters to 0, increase j by one, and go to Step 2.

The decoding procedure will stop if all the parity-checks are satisfied or if the limit on the maximum number of iterations is reached.

It is well known that this iterative decoding algorithm used with LDPC codes has an error-correction capability that increases linearly with the length of the code. MDPC codes have slightly higher parity-check weight than LDPC codes and this slightly influences the error-correction capability in a negative way. So the actual performance of this algorithm on MDPC codes is relatively poor compared with that on LDPC codes. More details on the decoding performance can be found in [Mis+13]. Other variants of this decoding algorithm have been proposed [HMG13; MOG15], to reduce the decoding error probability, involving changing the flipping and the thresholds, introducing more rounds, or other techniques. The error probability for proposed parameters is still large, for any decoding algorithm, compared to the corresponding security level. For 80-bit security, it is typically in the range $10^{-4} - 10^{-8}$, whereas a value of 2^{-80} would be required to be more sure that decoding errors would not be a tool in cryptanalysis.

2.5 Proposed Parameters

In [Mis+13], the following parameters were proposed for a QC-MDPC scheme with 80-bit, 128-bit and 256-bit security level.

Table 1: Proposed QC-MDPC instances with key size and security level.

Parameters						
n	r	w	t	n_0	Key size	Security
9602	4801	90	84	2	4801	80
19714	9857	142	134	2	9857	128
65542	32771	274	264	2	32771	256

Implementations of the QC-MDPC scheme [HMG13; MOG15] and a variant [MHG16] demonstrate excellent efficiency in terms of computational complexity and key sizes for encryption and decryption on constrained platforms such as embedded micro-controllers and FPGAs.

Key Recovery Attack

A major attack on the QC-MDPC scheme was recently described in [GJS16]. In the underlying setting, it uses the assumption that Mallory, taking the role of Alice, can observe the reaction of Bob, i.e. whether the decryption step was successful or if there was a decoding error. This is a natural assumption, as if Bob received a decoding error he would typically need to ask for a retransmission of the message in some way.

The attack in [GJS16] recovers the secret key. The objective of the key recovery attack is to recover the parity check matrix H (knowing only G). Obviously, H can be easily be derived from any H_i, thereby reducing the problem to finding H_0. Since H_0 is a circular matrix, it is enough to recover the first row h_0.

The key idea is to examine the decoding result for different error patterns. In particular, Mallory will pick errors from special subsets. Let Ψ_d be the set of all binary vectors of length $n = 2r$ having exactly t ones, where all ones are placed with distance d in the first half of the vector. The second half of the vector is all zero. The set Ψ_d guarantees repeated ones at distance d at least $t/2$ times, where

$$\begin{aligned}\Psi_d \;=\; & \{\mathbf{v} = (\mathbf{e}, \mathbf{f}) \mid w_H(\mathbf{f}) = 0, \text{ and } \exists \text{ distinct } s_1, s_2, \ldots, s_t, \text{ s.t. } \mathbf{e}_{s_i} = 1, \text{ and} \\ & s_{2i} = (s_{2i-1} + d) \bmod r \text{ for } i = 1, \ldots, \frac{t}{2}, \text{ and } w_H(\mathbf{e}) = t\}.\end{aligned}$$

Mallory will now send M messages to Bob, using QC-MDPC with the error selected from the subset Ψ_d. When there is a decoding error with Bob, she will record this and after M messages she will be able to compute an empirical decoding error probability for the subset Ψ_d. Furthermore she will do this for $d = 1, 2, \ldots, U$ for some suitable upper bound U.

Algorithm 1 Computing the distance spectrum

```
Input: parameters $n, r, w$ and $t$ of the underlying QC-MDPC code, number of
    decoding trials $M$ per distance.
Output: distance spectrum $D(\mathbf{h}_0)$ (multiplicity vector).
for all distances $d$ do
    Try $M$ decoding trials using the designed error pattern
    Perform statistical test to decide multiplicity $\mu(d)$
    Set position $d$ in $D(\mathbf{h}_0)$ to the multiplicity $\mu(d)$
end for
```

The main observation is that there is a strong correlation between the decoding error probability for error vectors from Ψ_d and the existence of a distance d between two ones in the secret vector $\mathbf{h}_0$. If there exists two ones in $\mathbf{h}_0$ at distance d, the decoding error probability is much smaller than if distance d does not exist between two ones.

After sending $M \times U$ messages, we look at the decoding error probability in each Ψ_d and classify each d, $d = 1, 2, \ldots, U$ according to its multiplicity, since each distance can appear many times. We denote the multiplicity as $\mu(d)$. This provides a distance spectrum for $\mathbf{h}_0$, which we denote as $D(\mathbf{h}_0)$. In this paper the distance spectrum is presented as a *multiplicity vector* of length U (as opposed to a set, which was used in [GJS16]) and defined as

$$D(\mathbf{h}_0) = (\mu(1), \mu(2), \ldots, \mu(U)).$$

As an example from [GJS16], for the bit pattern $\mathbf{c} = 0011001$ we have $U = 3$ and

$$D(\mathbf{c}) = (1, 0, 2),$$

with distance multiplicities $\mu(1) = 1, \mu(2) = 0$ and $\mu(3) = 2$.

The procedure for computing the distance spectrum is specified in Algorithm 1.

The final step is to do derive $\mathbf{h}_0$ from knowing the distance spectrum $D(\mathbf{h}_0)$. This is rather straightforward. Start by assigning the first two ones in a length i_0 vector in position 0 and i_0, where i_0 is the index of the smallest non-zero value in the empirical $D(\mathbf{h}_0)$. Then put the third one in a position and test if the two distances between this third one and the previous two ones both appear in the distance spectrum. If they do not, we test the next position for the third bit. If they do, we move to test the fourth bit and its distances to the previous three ones, etc. After reconstruction, we have restored $\mathbf{h}_0$. Reconstruction is possible even if some smaller fraction of entries in the empirical $D(\mathbf{h}_0)$ are wrong and/or are missing. We refer to [GJS16] for more details on the reconstruction part.

The attack was implemented and tested on a proposed instance of QC-MDPC for 80-bit security. It used about 240M ciphertexts (in the CPA-attack scenario) and successfully recovered the secret key with low computational complexity.

3 A New Improved Attack through a Chaining Method for Error Vectors

The main contribution of this paper is to propose a new adaptive way of selecting error vectors (patterns) which provides an improved attack for the CPA scenario. We introduce this new chaining method as a way to artificially increase the decoding error rate for codes that use iterative decoders. The chaining method works by leveraging the knowledge of a single error pattern into finding a new pattern that is similar to the first pattern. By repeating the process a chain of similar error patterns is created.

This section will provide a detailed description of both the chaining method and how it might be used as an attack on the QC-MDPC scheme. We will also discuss how side channel information might be used to improve the efficiency of the attack. But first the attack models used in this paper are presented.

3.1 Attack Models

The following attack scenario is assumed. Bob continuously receives messages, encrypted with his public key, from Mallory. Mallory is able to detect each time Bob fails to decrypt any of the messages sent by her. By crafting the messages sent to Bob in a certain way and doing it a number of times it is possible for Mallory to recover Bob's secret key. This is a so called *reaction attack*, it is similar in nature to an *adaptive chosen ciphertext attack* (CCA2), but requires only to know decryption success or failure, not the result of the decryption. Thus, this attack model requires weaker assumptions than CCA2, but stronger than CPA (*chosen plaintext attack*). This scenario is very similar to the one described by Guo et al. in [GJS16], although the attacks described here only targets the case when the error can be freely chosen by Mallory. For definitions on CPA, CCA and CC2 we refer the reader to [Smal6].

A version of the new attack, where side-channel information is used is also included. This attack version requires an attack model which additionally allows timing information to leak about the decoding such that the number of iterations can be determined. This would be provided by a decoder implementation if it is not running in constant time.

3.2 Description of the Basic Attack

The chaining method for generation of error patterns works by first finding an initial error pattern $\mathbf{e}_0$, that fails to be decoded. We do this by random selection and trial until one is found. It will be shown later that the finding of $\mathbf{e}_0$ can be improved upon by utilizing side-channel information (see Section 3.5).

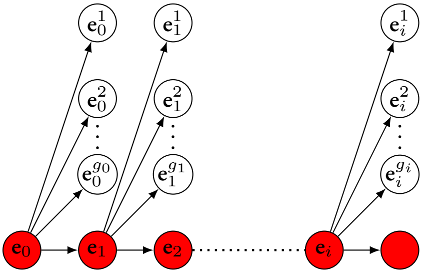


Figure 1: Visualization of error chains. White and red nodes indicate error patterns that can and cannot be decoded, respectively. Starting from an undecodable error pattern $\mathbf{e}_0$, we successively generate several new error patterns $\mathbf{e}_0^g$ by swapping a randomly chosen 0 and 1 in $\mathbf{e}_0$. Once a new undecodable error pattern $\mathbf{e}_1$ is found, it is used as the new base for generating new error patterns. By iterating this procedure, we end up with a (red) chain of undecodable error patterns.

Let an error pattern $\mathbf{e}_i$ be given, causing a decoding error. It is known that if a particular distance d in the distance spectrum $D(\mathbf{e}_i)$ does *not* exist in the key represented by the secret vector $D(\mathbf{h}_0)$ (e.g. the value of position d in $D(\mathbf{h}_0)$ is zero), then the probability of decoding failure is increased [GJS16]. This means that if we, somehow, can find another similar error pattern, denoted $\mathbf{e}_{i+1}$, that also fails to be decoded, then the *differences* between the distance spectra of these two error patterns should also contribute to error patters which are harder to decode than the average. We can detect this by measuring an increased decoding failure rate for such patterns.

We use the following notation. A distance spectrum of a binary vector $\mathbf{e}$ of length $2U$ is written as $D(\mathbf{e}) = (D_1(\mathbf{e}), D_2(\mathbf{e}), \ldots, D_U(\mathbf{e}))$. The difference between two spectra for vectors $\mathbf{e}_i$, $\mathbf{e}_{i+1}$ is defined as the element-wise subtraction of the two vectors, denoted $D(\mathbf{e}_i) - D(\mathbf{e}_{i+1})$. We simplify by writing for each distance d, where $1 \leq d \leq U$,

$$\Delta D_d = D_d(\mathbf{e}_i) - D_d(\mathbf{e}_{i+1})$$

In short ΔD_d indicates the change in the multiplicity for the specific distance d, when comparing $D(\mathbf{e}_{i+1})$ to $D(\mathbf{e}_i)$.

It is possible to find an error pattern similar to $\mathbf{e}_i$ by randomly[1] bit-permuting a copy of $\mathbf{e}_i$. In this copy, denoted $\mathbf{e}'$, both a single bit position containing the value 1 and a single bit position with the value 0 are randomly selected. These bits are then flipped (effectively moving a single bit in the pattern). Afterwards, the new error pattern is used in the encryption and the corresponding ciphertext is subsequently sent to Bob. If $\mathbf{e}'$ can be successfully decoded (CASE-0) then it is discarded and we create a new $\mathbf{e}'$ by again modifying a copy of $\mathbf{e}_i$. If the decoding fails (CASE-1) then $\mathbf{e}_{i+1} \leftarrow \mathbf{e}'$ and the bit-permutation procedure is repeated with $\mathbf{e}_{i+1}$ instead being the new base error pattern (and we increment i). See Fig. 1 and Algorithm 2.

For CASE-1, a distance d is less likely to be added (e.g. $D_d(\mathbf{e}_{i+1}) > D_d(\mathbf{e}_i)$) if d also exists as a distance in h_0. We introduce the vector F to represent the aggregated distance spectrum of all additions to the spectrum (i.e. $\Delta D_d > 0$). Similarly the vector G represents the combined distance spectrum of all distance removals from the patterns in the chain. Running through all differences between consecutive error patterns $\mathbf{e}_{i+1}, \mathbf{e}_i$ in the created chain, for $i = 1, 2, \ldots$, we update F and G as

$$\begin{cases} F_d \leftarrow F_d + \Delta D_d & \textbf{if } \Delta D_d > 0, \\ G_d \leftarrow G_d - \Delta D_d & \textbf{if } \Delta D_d < 0, \\ \text{no change} & \textbf{if } \Delta D_d = 0, \end{cases} \qquad \forall d \in \{1, \ldots, U\}, \tag{1}$$

[1] We wish to remark that there may be other methods of finding modifications that might be even more effective at amplifying the error rate. Search heuristics such as evolutionary algorithms or other methods are applicable here.

By combining the results of M decoding attempts (ciphertexts) the vectors F and G accumulate all changes to the distance spectra, which in turn gives us combined distance spectra of statistical significance (provided M is large enough).

For CASE-0 our simulations show that it is not a simple matter of reversing the logic above; Considering that $\mathbf{e}'$ now leads to successful decoding, additions (in respect to $\mathbf{e}_i$) to the distance spectrum could either be a large contributor towards the decoding success or a small one. A large contribution would indicate a higher probability of being represented in the distance spectrum of $\mathbf{h}_0$ and a small contribution would indicate a higher probability of *not* being represented. It will be shown in later sections that the number of iterations required for decoding plays a large role of how to interpret distance additions and deletions. The basic attack, as described in this section, ignore this and simply perform the same operations described above for CASE-1, although with different output vectors and different input. The outputs are stored into vectors A and B and for input we use those error pattern which resulted in decoding successes, instead of failures. So for all $\mathbf{e}'$ with decoding success we consider $\Delta D = D(\mathbf{e}_i) - D(\mathbf{e}')$ and update vectors A and B as

$$\begin{cases} A_d \leftarrow A_d + \Delta D_d & \textbf{if } \Delta D_d > 0, \\ B_d \leftarrow A_d - \Delta D_d & \textbf{if } \Delta D_d < 0, \\ \text{no change} & \textbf{if } \Delta D_d = 0, \end{cases} \qquad \forall d \in \{1, \ldots, U\}. \tag{2}$$

3.3 The attack when Side-Channel Information is available

Let us consider the scenario where Bob is not using a constant time implementation. We assume that this will give us information on how many iterations that were used in the decoding step. In such a scenario it is possible to more efficiently utilize CASE-0 information when mounting an attack, increasing the amplification effect of the chaining method even further.

In practice, that attack is performed in a very similar manner to the one without side-channel information. We simply modify it so that we additionally save the difference of the distance spectra for each successful decoding (CASE-0) into *different* vectors depending on the number of iterations required for the decoding. Introducing vectors $A_j = (A_{1,j}, A_{2,j}, \ldots, A_{U,j})$ and similarly vectors B_j, where j runs through the possible values for the number of iterations, we have an update of the form

$$\begin{cases} A_{d,j} \leftarrow A_{d,j} + \Delta D_d & \textbf{if } \Delta D_d > 0, \\ B_{d,j} \leftarrow B_{d,j} - \Delta D_d & \textbf{if } \Delta D_d < 0, \\ \text{no change} & \textbf{if } \Delta D_d = 0, \end{cases} \qquad \forall d \in \{1, \ldots, U\}. \tag{3}$$

Algorithm 2 Algorithm for constructing error pattern chains and gathering distance information, without side channel information.

```
Input: $\mathbf{e}_0, T, m, G$    ▷ $\mathbf{e}_0$: initial error pattern, T: chain length
Output: $A, B, G, F$
$A \leftarrow$ zero-vector of length $r/2$
$B \leftarrow$ zero-vector of length $r/2$
$F \leftarrow$ zero-vector of length $r/2$
$G \leftarrow$ zero-vector of length $r/2$
$\mathbf{e} \leftarrow \mathbf{e}_0$
$i \leftarrow 0$
while $i < T$ do
    $s \leftarrow$ distance spectrum of $\mathbf{e}$
    repeat
        $x \leftarrow$ position of a random 1 in $\mathbf{e}$
        $y \leftarrow$ position of a random 0 in $\mathbf{e}$
        $\mathbf{e}' \leftarrow$ copy $\mathbf{e}$
        flip bits $x$ and $y$ in $\mathbf{e}'$    ▷ Random 1-bit permutation of $\mathbf{e}$
        $s' \leftarrow$ distance spectrum of $\mathbf{e}'$
        $\mathbf{c} \leftarrow mG + \mathbf{e}'$    ▷ Encrypt with $\mathbf{e}'$
        $l \leftarrow$ is $\mathbf{c}$ decodable?    ▷ Attempt to decrypt $\mathbf{c}$
        for all indexes $d$ of $s$ do    ▷ Loop through distance spectrum (DS)
            $j \leftarrow s'[d] - s[d]$    ▷ Value of $\Delta D_d$
            if $j > 0$ then
                if $l$ then
                    $F[d] \leftarrow F[d] + |j|$    ▷ Save distance additions for decoding successes
                else
                    $A[d] \leftarrow A[d] + |j|$    ▷ Save distance additions for decoding failures
                end if
            else if $j < 0$ then
                if $l$ then
                    $G[d] \leftarrow G[d] + |j|$    ▷ Save distance deletions for decoding successes
                else
                    $B[d] \leftarrow B[d] + |j|$    ▷ Save distance deletions for decoding failures
                end if
            end if
        end for
    until not $l$
    $e \leftarrow \mathbf{e}'$
    $i \leftarrow i + 1$
end while
```

As was explained in the previous section, the number of iterations influences which distances are more probable to appear or disappear, when comparing $D(\mathbf{e}_i)$ and $D(\mathbf{e}')$. Now j is the number of iterations required for decoding error pattern $\mathbf{e}'$. For a specific implementation of the decoder, there exists a value I where the probabilities flip. By this we mean that if $j < I$ our simulations show that distances d that exist in $D(\mathbf{h}_0)$ are more probable to be represented in $A_{d,j}$. Conversely, if $j \geq I$, the same distances are *less* likely to be represented in $A_{d,j}$.

The simulations show that for $B_{d,j}$ there is no such property. All distances d, regardless of j, are less likely to disappear from $D(\mathbf{e}')$ if they exist in h_0 (although the statistical significance varies with both the implementation of the decoder and with j).

3.4 Performing an attack

Using vectors F, G, A and B, or optionally $A_j, B_j, \forall j$ in place for plain A and B, one may reconstruct the distance spectrum of h_0 with relative ease, provided the number of samples M is large enough.

An intuitive explanation of the statistical nature of our experiment can be given by using an alternative representation of the vectors: Divide each element of the vectors F and G with the number of unsuccessful decoding attempts. The same is done for A_j and B_j (for each value of j), dividing with the number of successful decoding attempts which correspond to the value of j. If this is done then the results would be the probability for each d causing the underlying event (for example, for the F vector, this is the event that a distance d is added to the new error pattern and it causes a new decoding error).

A straightforward attack would be to simply use, for example, vector F and directly perform a key recovery using the same algorithm as published by Guo et al. However, to extract as much information as possible from each decoding attempt the following formula may be used to calculate a new aggregated distance vector (considering the case where side-channel information is available);

$$F + G + \left(\sum_{j=1}^{I-1} A'_j + B_j \right) + \left(\sum_{j=I}^{J} A_j + B_j \right), \tag{4}$$

where

$$A'_j = \max\left(A_j\right) - A_j + \min\left(A_j\right). \tag{5}$$

Here J is the maximum number of iterations allowed by the decoder implementation. The reason we calculate A'_j for $j < I$ instead of using A_j directly stems from the fact that the probability relationship for each distance d to occur in A_j is flipped when comparing with $A_j, j \geq I$. This follows directly from the discussion given in the previous section. We calculate A'_j in this manner to preserve the total sum of the vector so that the weighting of each vector is proportional to the number of samples used to build each vector.

If side-channel information is *not* available, then the above formula might be simplified as

$$F + G + A' + B, \tag{6}$$

where

$$A' = \max(A) - A + \min(A). \tag{7}$$

We use A' here for the same reasons as when we are calculating with Eq. (4), except here we are assuming that the number of ciphertexts resulting in a j below I is greater than the number of ciphertexts resulting in $j \geq I$. This has always been the case in our simulations and it can be checked by the reader by comparing the ratio of each iteration in Fig. 2b with the I derived from Fig. 5a. As long as the assumption holds true the vector A will result in an aggregated distance spectrum with inverse probabilities compared to, for example, the F vector. The results added to A vector where $j \geq I$ will add noise, which is why using side-channel information provides for a more effective attack.

Eqs. (4) and (6) are simple formulas, and it is conceivable that they might be further optimized, for example by introducing some weight factor for each vector. This is not done in this paper and may be regarded as future work.

3.5 Speeding up $\mathbf{e}_0$ generation

The algorithm described thus far requires $\mathbf{e}_0$ (a first undecodable error pattern) as input. Depending on the decoding failure rate of the decoder being used this might not be trivial in practice. As it happens, one might actually utilize the side-channel attack described above to find the first undecodable error pattern.

This is done by selecting any random error pattern, attempt to decode it, store the number of iterations required for decoding and modify the pattern similarly to how we do it in chaining method described above. Then we try to decode the new pattern, if the number of iterations are lower we discard the modifications and try again, otherwise we use the modified error pattern as the new point-of-origin and base the modifications on this new pattern. We keep this up until a pattern has been found that cannot be decoded. Our simulations show this method offers significant speedups compared to the standard method of random trial and error.

4 Implementations and Numerical Results

In this section we describe how our simulations[2] were made and what results were produced. First we describe how the simulations were designed. Next, we describe the different decoder implementations used in this paper and how they affect the decoding failure rate. We follow up on this by showing the amplification effect on these different implementations, according to our simulation results. Then we

[2] Source code available upon request.

show, using our simulations on the QC-MDPC scheme, that the amount of necessary ciphertexts are indeed significantly reduced compared to the original attack by Guo et al. Finally, the effect of using side-channel information is analyzed.

4.1 Simulation set-up

All results shown in this paper are based on simulations using the same key and using parameters for 80-bit security. We have confirmed our findings by rerunning our simulations using 9 other keys, although those results are not shown here, for sake of brevity.

During these simulations we make use of the chaining method to find $\mathbf{e}_0$, as described in Section 3.5. We confirm with our simulations that using side-channel information gives significant speedup. In fact, even though we are using decoder $\mathcal{Q}$ (see below) we can find $\mathbf{e}_0$ in a matter of seconds or a few minutes at most.

4.2 Decoder implementations

The decoder implementations used in this paper are based on the descriptions of $\mathcal{B}$ and $\mathcal{F}$, given in [HMG13]. We have also implemented a decoder $\mathcal{Q}$ based on the descriptions provided in [Cho16].

Decoder $\mathcal{B}$

This is the standard Gallager probabilistic iterative bit-flipping decoder implemented as originally described in [Gal62] and later in [HMG13]. Once per iteration the syndrome is computed to determine how many parity checks are violated. This decoding algorithm is using precomputed threshold values T_j, where j is the current iteration number. These are used to determine what bits violate at least T_j parity checks and those that do are flipped. Before starting the next iteration the syndrome is compared to zero. If it is zero then the algorithm stops. For this algorithm the maximum number of iterations J is set to 10, and once $j > J$ the decoder stops and returns a decoding failure. The thresholds, for $\mathcal{B}$, are (as given in [HMG13]):

$$28, 26, 24, 22, 20, 18, 16, 14, 12, 10$$

This algorithm is not constant-time, and extracting side-channel information would be trivial. In our experiment we have modified it to return the actual number of iterations and thus removing the noise that would have been introduced by measuring the amount of time required for decoding (as would be required in a real attack scenario).

As we can see in Fig. 2a this decoder has a relatively high decoding failure rate and there are other implementations that perform better in this regard, as is shown in the next two sections. This is the decoder that was used in [GJS16].

Decoder $\mathcal{F}$

This decoder is a variant of decoder $\mathcal{B}$ and is also implemented as described in [HMG13]. Instead of computing the syndrome once every iteration however, it *directly* updates the syndrome as soon as a bit is flipped (this is called an *in-place* decoder). It also compares the syndrome to zero after each update and exits before the current iteration has run its course. It uses the same T_j and J as decoder $\mathcal{B}$. As can be seen in Fig. 2a this strategy improves the decoding failure rate, compared to decoder $\mathcal{B}$.

Decoder $\mathcal{Q}$

This decoder is our implementation of *QcBits* (pronounced "quick bits"). We have implemented it according to the description provided in [Cho16]. Our implementation does not implement any of the performance enhancements nor does it attempt to be constant time. Algorithmically, *QcBits* appears to be equivalent with decoder $\mathcal{F}$ but with different threshold values T_j and $J = 6$. It should be noted however that our implementation $\mathcal{Q}$ uses $J = 10$, since keeping J constant across all implementations makes comparisons of the decoder characteristics easier. The thresholds used are, as given in [Cho16] and the published source code of *QcBits*:

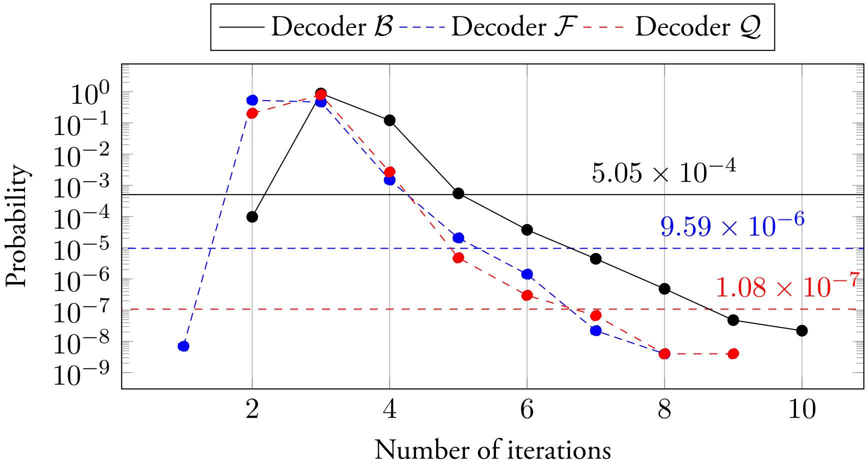
$$29, 27, 25, 24, 23, 23, 23, 23, 23, 23$$

It is worth noting that the last 4 values are *not* given in [Cho16], since they stop at 6 iterations, but were instead found in the published source code. As can be seen in Fig. 2a these 4 last values are indeed not optimal and should probably instead follow a decreasing formula in the same manner as for decoders $\mathcal{B}$ and $\mathcal{F}$.

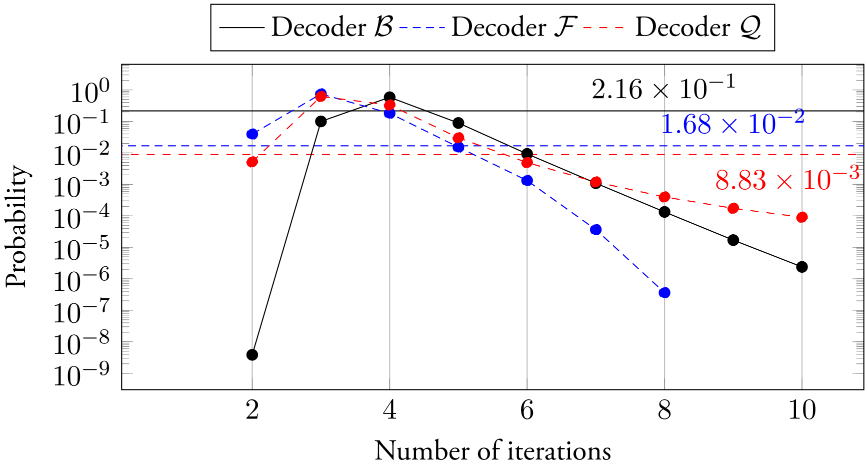
The author of *QcBits* claims no decoding failures when decoding 10^8 ciphertexts. As can be seen in Fig. 2a we were unable to reproduce this very low decoding failure rate. Our implementation might be flawed, but regardless, decoder $\mathcal{Q}$ shows value to us in our simulations due to its decoding failure rate still beating the other decoders. This enables us to get a good indication of the amplification effect of the chaining method, as we will see in the next section.

4.3 Amplification Effect

To test the amplification effect of the chaining method we implemented a small test routine that generates ciphertexts with error patterns according to our chaining method. We collected 2^{28} number of samples by running the algorithm in 20 threads and thereby creating 20 separate error chains. The entire test was run 10 times (with different keys) for each algorithm. As can be seen in Fig. 2b the amplification effect is significant regardless of the decoding failure rate of the decoder in question. In fact, based on the results of these simulations, the amplification effect



(a) Without error amplification (uniformly random error patterns). This data is based on 2^{28} number of decoding trials each, for decoders $\mathcal{B}, \mathcal{F}$ and $\mathcal{Q}$.

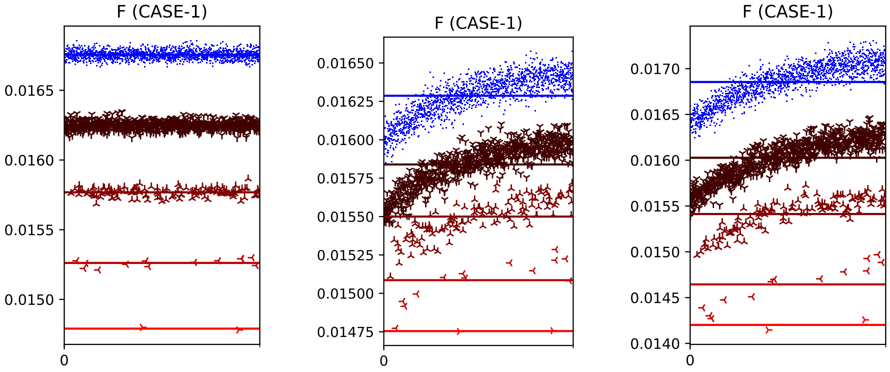


(b) With error amplification (chaining method). This data is based on 2^{28} number of decoding trials each, for decoders $\mathcal{B}, \mathcal{F}$ and $\mathcal{Q}$.

Figure 2: In these logarithmic plots the markers indicate by what probability (y-axis) each of the implementations can decode a message using (exactly) a certain number of iterations (x-axis). The horizontal lines indicate the decoder failure rate for each decoder implementation.

appears to increase for decoders with lower decoding failure rate. In these simulations we have only concerned ourselves with H_0. In practice, for 80-bit security, this means that we are only looking at the first half of the 9602-bit ciphertext.

4.4 Chaining Attack



(a) Decoder $\mathcal{B}$ (b) Decoder $\mathcal{F}$ (c) Decoder $\mathcal{Q}$

Figure 3: The plot for vector F for decoders $\mathcal{B}$, $\mathcal{F}$ and $\mathcal{Q}$. Multiplicity levels for each distance $d \in D(h_0)$ are easily distinguished. This data is based on 2^{28} decoding attempts, for each decoder. Blue dots indicate distances that do *not* appear in $\mathbf{h}_0$ (multiplicity 0). Dark markers indicate those distances that appear in $\mathbf{h}_0$ once (multiplicity 1). Higher multiplicities are marked analogously with less dark shades of red and with different markers. The horizontal lines represent the average probability, for each multiplicity. In [Eat+18] Eaton et al. discovered and discussed the asymmetric nature of *in-place* decoders. As we can see this asymmetric shape is also found in Fig. 3b and Fig. 3c.

Using Algorithm 2 as presented in Section 3.2 we generated the results shown in Fig. 3. It is worth noting here that plots of vectors F and G look almost identical (not shown here), and the underlying data is extremely similar (although not identical). These results show that the amount of extra information that can be extracted from using both vectors, instead of only one of them, is extremely limited.

A simple explanation of why this would be the case is given if one first considers the fact that each pattern is part of a chain. As such, a distance in the pattern cannot reasonably be removed from the distance spectrum of the current pattern $\mathbf{e}_i$ *unless* it has first been added to a pattern $\mathbf{e}_l$, where $l < i$.

From vector F in Fig. 3 we can easily see that the probability for each of the distances are stratifying based on the multiplicity of the distance in $\mathbf{h}_0$, just as described in Section 3.2 and in [GJS16]. It is interesting to note that the multiplicities are are not separating into layers by an equal amount, for the different decoder implementations. In Fig. 3 we see as expected that decoder $\mathcal{B}$ requires the least amount of ciphertexts in order to separate into different layers. However it appears that decoder $\mathcal{F}$ unexpectedly requires *more* ciphertexts than $\mathcal{Q}$. This is an interesting discovery that we have not investigated further.

Also noteworthy in Fig. 3 is the asymmetry of the results for the *in-place* decoders. It was, to our knowledge, first discovered and discussed by Eaton et al. in [Eat+18]. In order to utilize the results of these decoders in a real attack one would first be required to normalize the vectors according to a regression fitting, before categorizing each distance by its correct multiplicity.

In order to facilitate the comparison of our results with those of [GJS16], we will henceforth only consider results from simulations with decoder $\mathcal{B}$. In Fig. 4 we plot the number of errors on the accumulated distance spectrum as a function of the number of ciphertexts.

To calculate the number of errors we use knowledge of the key to partition each distance according to its multiplicity in the real key and to calculate the average of all multiplicities (see Fig. 3). For each distance d in the aggregated distance spectrum we find the multiplicity that is the closest and use this as our guess. Again we use knowledge of the key to detect if our guess is correct or wrong.

In Fig. 4 we see that the required number of ciphertexts is in the order of 12M as compared to the 240M in [GJS16] for CPA security, a speedup of a factor of 20. Since it was noted in Section 4.3 that the amplification effect appears to be increasing with the use of "better" decoder implementations we expect the speedup factor to increase in proportion. However, [GJS16] does not provide any such numbers for us to compare with.

In [GJS] it was shown that it is in fact not necessary to have a 100% correct distance spectrum in order to mount an attack. Guo et al. showed that 900 errors in the distance spectrum can be reliably tolerated (for 80-bit security) and still being able to successfully recover H in a reasonable time. For comparison, Guo et al. required 4M ciphertexts to recover a distance spectrum with 900 errors. Our simulations show this requirement to have dropped to approximately 310 000 ciphertexts, for decoder $\mathcal{B}$, using our amplification attack with the chaining method.

4.5 Chaining Attack, With Side-Channel Information

As was explained in Section 3.3, for vector A the probabilities for each distance d flips depending on if the number of iterations required to decode the ciphertext is below a implementation dependent threshold value I, or not. By close examination of Fig. 5a one can see that $I_{\mathcal{B}}$ is 5. Although not shown here, we have similar results for the other decoders: $I_{\mathcal{F}} = 4$ and $I_{\mathcal{Q}} = 4$. Fig. 5b confirms that there is no such analogous I for vector B, i.e. distances that are removed from the distance spectrum of $\mathbf{e}_i$.

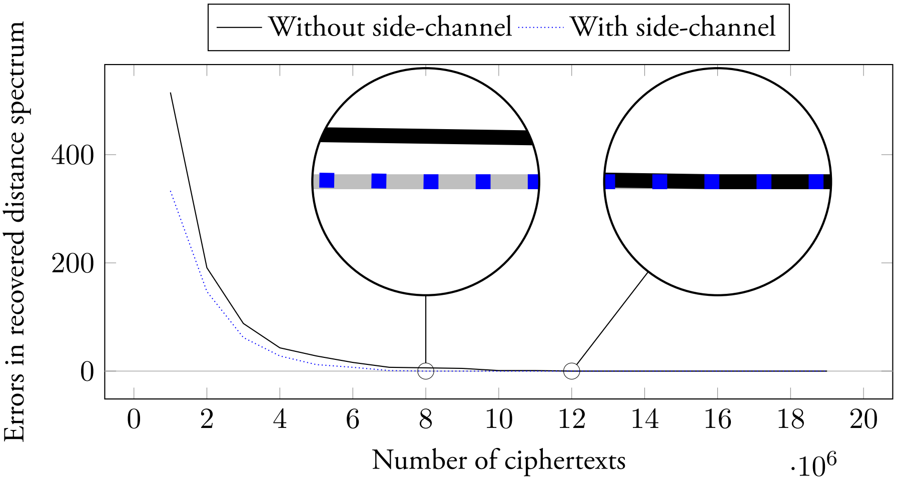
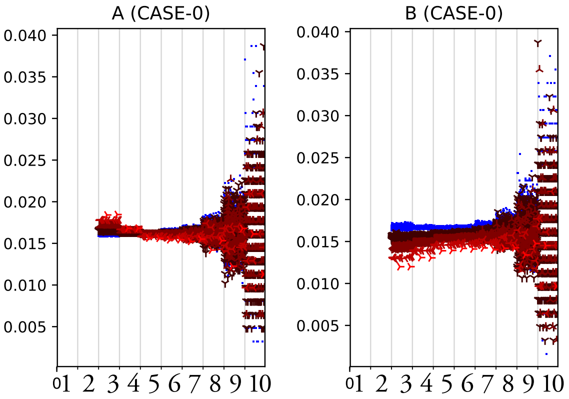


Figure 4: Number of errors in recovered distance spectrum for decoder $\mathcal{B}$. The number of errors is calculated according to the description given in Section 4.4. The results plotted are both with and without side-channel information. The circled values are the lowest number of ciphertexts needed where the number of errors reach zero (and stays there), with and without side-channel information respectively. For easier viewing $y = 0$ is shown as a grid line.

Fig. 4 show that the attack can indeed be further improved by utilizing side-channel information. A simple comparison with [GJS16] gives us a factor of $240/8 = 30$, (compared to 20 without side-channel information). We acknowledge that these calculations are based on ideal data which does not depend on timing measurements and as such provides no measurement errors nor any noise introduced by, for example, CPU scheduling and cache timings.

Non-constant time decoders

Fig. 6 presents the results of a simple experiment that measures the decoding time of 2^{16} ciphertext decodings, for each decoder. We argue that these results would indicate the practicality of using side channel information on non-constant time decoder implementations, in a close to real world setting. By observing the distributions of the measurements one can quickly see that the number of iterations are indeed easily distinguishable. It should be noted however that these measurements are done in-process and do therefore not introduce any additional noise, from for example, network latency.



(a) Plot of vector A_j where $0 < j \leq 10$. (b) Plot of vector B_j where $0 < j \leq 10$.

Figure 5: Decoder $\mathcal{B}$: The plot for vectors A and B. The vectors have been converted to show probabilities instead of raw values, for easier understanding of y-values. This data is based on 2^{28} ciphertexts. The plots show the probability for decoding success using j (x-axis) number of iterations for each distance in the distance spectrum. j in this plot is discrete and ranges from 1 to 10 and is indicated by the grid lines. The wider range of values for certain iterations (e.g. $j = 10$) are a by-product of the smaller sample size. In the Fig. 5a we can see that $I = 5$ since the probabilities flip for the results where $j \geq 5$.

5 Conclusions

In the general case, we have shown that the advertised decoding failure rate of a decoder implementation might not always tell the whole truth about the security of the particular implementation. We have shown that knowledge of a single error pattern might be used as leverage for an attacker to generate other difficult-to-decode error patterns. We get this amplification effect using the chaining method described in this paper.

Specifically for QC-MDPC we have shown that using the technique described in this paper one can mount a successful attack against the scheme using more than an order of magnitude fewer ciphertexts than previous state-of-the-art attacks, for CPA security. This is valid without using side-channels or private information such as syndrome weight. This is the result of both the error rate amplification and the fact that we can extract information also from decoding *successes* as well as failures.

Additionally we have shown that side-channel information can be used to improve the efficiency of the attack, if such information is available. It can also be used to speed up the discovery of the initial error pattern which we use to bootstrap the attack.

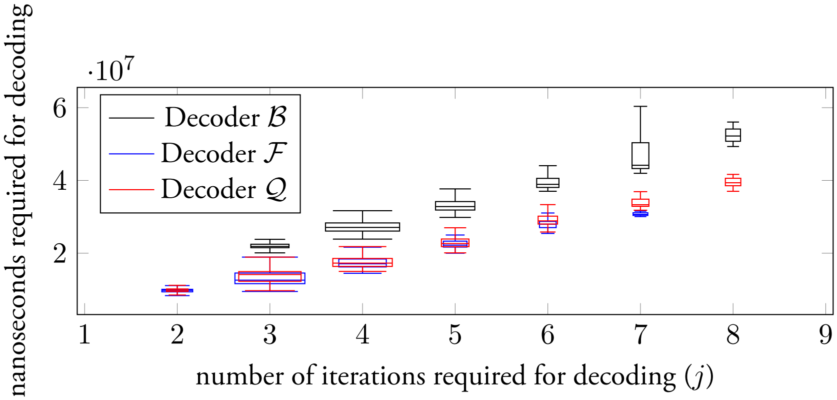


Figure 6: Box plots showing the measurement distribution of the number of nanoseconds (y-axis) for decoder $\mathcal{B}$, $\mathcal{F}$ and $\mathcal{Q}$ to decode a single message using j number of iterations (x-axis). The middle line shows the median and the upper and lower edges of the boxes show the upper and lower quartiles, respectively. The upper and lower whiskers in turn indicate the lowest and highest measurement still within 1.5 IQR (InterQuartile Range) of the upper and lower quartiles, respectively. The width indicate the relative number of collected measurements (i.e. the relative reliability) of each box plot. It should be noted that the underlying data has been cleaned from very obvious outliers before generating the plots. Roughly 2^{16} measurements, for each decoder, was used to generate the data which is summarized in this figure.

6 Further Work

In this paper we have only presented results related to parameters corresponding to 80 bits of security. It would be interesting to see how the results scale to 128- and 256-bit security parameters.

Another point of research would be to continue the simulations using other decoders based on different techniques. In [CS16], Chaulet et al. present a decoder which uses dynamically calculated thresholds, and which is able to achieve failure rates comparable to our decoder $\mathcal{Q}$.

Although Eaton et al. do give a tentative explanation in [Eat+18], the asymmetric nature of *in-place* decoders have not been fully explained to our satisfaction and we would like to have a more complete understanding of the effects of this asymmetry.

References

[Bal+07] M. Baldi, F. Chiaraluce, R. Garello, and F. Mininni. "Quasi-Cyclic Low-Density Parity-Check Codes in the McEliece Cryptosystem". In: *Proceedings of IEEE International Conference on Communications, ICC 2007, Glasgow, Scotland, 24-28 June 2007.* 2007, pp. 951–956.

[BBC08] M. Baldi, M. Bodrato, and F. Chiaraluce. "A New Analysis of the McEliece Cryptosystem Based on QC-LDPC Codes". In: *Security and Cryptography for Networks, 6th International Conference, SCN 2008, Amalfi, Italy, September 10-12, 2008. Proceedings*. 2008, pp. 246–262.

[Ber09] D. J. Bernstein. "Introduction to post-quantum cryptography". In: *Post-quantum cryptography*. Springer, 2009, pp. 1–14.

[Che+16] L. Chen et al. "Report on post-quantum cryptography". In: *National Institute of Standards and Technology Internal Report* 8105 (2016).

[Cho16] T. Chou. "QcBits: Constant-Time Small-Key Code-Based Cryptography". In: *Cryptographic Hardware and Embedded Systems – CHES 2016* (Jan. 1, 2016).

[CS16] J. Chaulet and N. Sendrier. "Worst case QC-MDPC decoder for McEliece cryptosystem". In: *CoRR* abs/1608.06080 (2016). arXiv: `1608.06080`.

[Eat+18] E. Eaton, M. Lequesne, A. Parent, and N. Sendrier. "QC-MDPC: A Timing Attack and a CCA2 KEM". In: *International Conference on Post-Quantum Cryptography*. Springer. 2018, pp. 47–76.

[Fab+17] T. Fabšič, V. Hromada, P. Stankovski, P. Zajac, Q. Guo, and T. Johansson. "A Reaction Attack on the QC-LDPC McEliece Cryptosystem". In: *International Workshop on Post-Quantum Cryptography*. Springer. 2017, pp. 51–68.

[FHZ18] T. Fabšic, V. Hromada, and P. Zajac. "A Reaction Attack on LEDApkc". In: (2018). Cryptology ePrint Archive, Report 2018/140 (2018). `http://eprint.iacr.org/`.

[Gal62] R. Gallager. "Low-density parity-check codes". In: *IRE Transactions on information theory* 8.1 (1962), pp. 21–28.

[GJS] Q. Guo, T. Johansson, and P. Stankovski Wagner. "A Key Recovery Reaction Attack on QC-MDPC". In: Accepted for publication in IEEE Transactions on Information Theory, final manuscript in preparation. Full version of [GJS16].

[GJS16] Q. Guo, T. Johansson, and P. Stankovski. "A Key Recovery Attack on MDPC with CCA Security Using Decoding Errors". eng. In: *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Proceedings*. Vol. 10031 LNCS. Springer Verlag, 2016, pp. 789–815.

[Gop70] V. D. Goppa. "A new class of linear correcting codes". In: *Problemy Peredachi Informatsii* 6.3 (1970), pp. 24–30.

[HMG13] S. Heyse, I. von Maurich, and T. Güneysu. "Smaller Keys for Code-Based Cryptography: QC-MDPC McEliece Implementations on Embedded Devices". In: *Cryptographic Hardware and Embedded Systems - CHES 2013: 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*. Ed. by G. Bertoni and J.-S. Coron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 273–292.

[LJ12] C. Löndahl and T. Johansson. "A New Version of McEliece PKC Based on Convolutional Codes". In: *Information and Communications Security - 14th International Conference, ICICS 2012, Hong Kong, China, October 29-31, 2012. Proceedings*. 2012, pp. 461–470.

[McE78] R. J. McEliece. "A Public-Key Cryptosystem Based On Algebraic Coding Theory". In: *DSN Progress Report 42–44* (1978), pp. 114–116.

[MHG16] I. von Maurich, L. Heberle, and T. Güneysu. "IND-CCA Secure Hybrid Encryption from QC-MDPC Niederreiter". In: *Post-Quantum Cryptography*. Springer, 2016, pp. 1–17.

[Mis+13] R. Misoczki, J.-P. Tillich, N. Sendrier, and P. S. Barreto. "MDPC-McEliece: New McEliece Variants from Moderate Density Parity-Check Codes". In: *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*. IEEE. 2013, pp. 2069–2073.

[MOG15] I. v. Maurich, T. Oder, and T. Güneysu. "Implementing QC-MDPC McEliece Encryption". In: *ACM Transactions on Embedded Computing Systems (TECS)* 14.3 (2015), p. 44.

[MS77] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error Correcting Codes*. Vol. 16. Elsevier, 1977.

[Sma16] N. P. Smart. *Cryptography Made Simple*. Information Security and Cryptography. Springer, Heidelberg, Germany, 2016.

Decryption Failure Attacks on IND-CCA Secure Lattice-Based Schemes

Abstract

In this paper we investigate the impact of decryption failures on the chosen-ciphertext security of lattice-based primitives. We discuss a generic framework for secret key recovery based on decryption failures and present an attack on the NIST Post-Quantum Proposal ss-ntru-pke. Our framework is split in three parts: First, we use a technique to increase the failure rate of lattice-based schemes called failure boosting. Based on this technique we investigate the minimal effort for an adversary to obtain a failure in three cases: when he has access to a quantum computer, when he mounts a multi-target attack or when he can only perform a limited number of oracle queries. Secondly, we examine the amount of information that an adversary can derive from failing ciphertexts. Finally, these techniques are combined in an overall analysis of the security of lattice based schemes under a decryption failure attack. We show that an attacker could significantly reduce the security of lattice based schemes that have a relatively high failure rate. However, for most of the NIST Post-Quantum Proposals, the number of required oracle queries is above practical limits. Furthermore, a new generic weak-key (multi-target) model on lattice-based schemes, which can be viewed as a variant of the previous framework, is proposed. This model further takes into consideration the weak-key phenomenon that a small fraction of keys can have much larger decoding error probability for ciphertexts with certain key-related properties. We apply

J.-P. D'Anvers, Q. Guo, T. Johansson, A. Nilsson, F. Vercauteren, and I. Verbauwhede. "Decryption Failure Attacks on IND-CCA Secure Lattice-Based Schemes". In: *Public-Key Cryptography – PKC 2019*. Ed. by D. Lin and K. Sako. Cham: Springer International Publishing, 2019, pp. 565–598 this model and present an attack in detail on the NIST Post-Quantum Proposal – ss-ntru-pke – with complexity below the claimed security level.

This paper is the result of a merge of [DVV18] and [GJN19].

1 Introduction

The position of integer factorization and the discrete logarithm problem as a cornerstone for asymmetric cryptography is being threatened by quantum computers, as their ability to efficiently solve these mathematical problems compromises the security of current asymmetric primitives. These developments have led to the emergence of post-quantum cryptography and motivated NIST to organize a post-quantum cryptography standardization process, with the goal of standardizing one or more quantum-resistant public-key cryptographic primitives. Submissions originate from various fields within post-quantum cryptography, such as lattice-based, code-based and multivariate cryptography.

Lattice-based cryptography has recently developed into one of the main research areas in post-quantum cryptography. Lattice-based submissions to the NIST Post-Quantum process can be broadly put into one of two categories: NTRU-based schemes (e.g. [Zha+17; Sch+17a]) and schemes based on the learning with errors (LWE) hard problem [Reg05]. A lot of research has been done on their security, such as provable post-quantum secure transformations from IND-CPA to IND-CCA secure schemes [HHK17; TU16; SXY17; Jia+17], security estimates of post-quantum primitives [APS15; Alb+18] and provable reductions for various hard problems underlying the schemes [Reg05; Pei09; LPR10; BPR12; Bra+13]

A striking observation is that numerous proposed Key Encapsulation Mechanisms (KEM's) have a small failure probability during decryption, in which the involved parties fail to derive a shared secret key. This is the case for the majority of schemes based on lattices, codes or Mersenne primes. The probability of such failure varies from 2^{-64} in Ramstake [Sze17] to 2^{-216} in New Hope [Sch+17c], with most of the failure probabilities lying around 2^{-128}. As this failure is dependent on the secret key, it might leak secret information to an adversary. However, reducing this probability has a price, as the parameters should be adjusted accordingly, resulting in a performance loss. An approach used by some schemes is to use error-correcting codes to decrease the failure probability. This leads to a reduction in the communication overhead, but makes the scheme more vulnurable to side-channel attacks.

As suggested by the wide range of failure probabilities in the NIST submissions, the implications of failures are still not well understood. In the absence of a clear evaluation technique for the impact of the failure rate, most NIST submissions have chosen a bound on the decryption failure probability around 2^{-128} based on educated guessing. As far as we know, only NIST-submission Kyber [Sch+17b] provides an intuitive reasoning for its security against decryption failure attacks, but this approximation is not tight. They introduce a methodology that uses Grover's search algorithm to find ciphertexts that have a relatively high probability of triggering a decryption failure.

1.1 Related Works

The idea of exploiting decryption errors has been around for a long time and applies to all areas of cryptography [Bol+14]. For lattice-based encryption systems, the Ajtai-Dwork scheme and NTRU have been a target for attacks using decryption failures. Hall, Goldberg, and Schneier [HGS99] developed a reaction attack which recovers the Ajtai-Dwork private key by observing decryption failures. Hoffstein and Silverman [HS00] adapted the attack to NTRU and suggested modifying NTRU to use the Fujisaki-Okamoto transform [FO99] to protect against such attacks. Further work in this direction is given in [JJ00], [How+03a] and [GN07]. Fluhrer [Flu16] described an attack against Ring-Learning with Errors (RLWE) schemes. In [Din+16] his work was extended to more protocols and in [Ber+18] a chosen-ciphertext attack on the proposal HILA5 [Saa17] was given, using decryption failures.

These attacks are chosen-ciphertext attacks on proposals with only IND-CPA-security and can be thwarted using an appropriate transformation to a chosen-ciphertext secure scheme, such as the Fujisaki-Okamoto transformation [FO99]. Hofheinz et al. [HHK17] and later Jiang et al. [Jia+17] proved a bound on the impact of the failure rate on an IND-CCA secure KEM that is constructed using this transformation, but their bounds are squared in the failure probability in the quantum oracle setting, which seems a very conservative result. Guo, Johansson and Stankovski [GJS16] proposed a key-recovery attack against the IND-CCA-secure version of QC-MDPC, which is a code-based scheme. It uses a distinguishing property that "colliding pairs" in the noise and the secret can change the decryption failure rate.

1.2 Contributions

In this paper we investigate the requirements for KEM's to resist decryption failure cryptanalysis. Having better security estimates can benefit the parameter selection process, resulting in improved security and efficiency. We focus on IND-CCA secure KEM's based on the (Ring/Module) Learning with Errors and (Ring/Module) Learning with Rounding paradigms. Nonetheless, the general method can also be applied to investigate the impact of failures on other schemes.

The exploitation of decryption failures of an IND-CCA secure cryptographic scheme proceeds in two main steps: obtaining ciphertexts that result in a decryption failure and estimating the secret based on these ciphertexts. In the first step, an adversary can use failure boosting to find 'weak' input vectors that artificially enlarge the failure rate of the scheme. In Section 3, we examine how an adversary can generate these 'weak' ciphertexts that increase the failure probability. We provide a theoretical framework and a Python implementation[1] to calculate an estimate of the minimum effort required for an adversary to obtain one failing ciphertext.

Once ciphertexts that trigger a decryption failure are collected, they can be used to estimate the secret. In Section 4, we study how much information is leaked by the collected failures. We develop a statistical model to estimate the secret from the failures and determine the residual entropy of the secret after a certain number of failures is collected. The estimate of the secret can be used to construct an easier problem that can be solved faster.

Section 5 combines failure boosting and the secret estimation technique from Section 4 to estimate the security of schemes based on (Ring/Module) Learning with Errors and (Ring/Module) Learning with Rounding under an attack exploiting decryption failures. We show that an attacker could significantly reduce the security of some schemes if he is able to perform sufficient decryption queries. However, for most NIST submissions, the number of decryption queries required is above practical limits.

In Section 6 we propose a new generic weak-key (multi-target) model exploiting the fact that a fraction of keys employed can have much higher error probability if the chosen weak ciphertexts satisfy certain key-related properties. The detailed attack procedure is similar to the attack discussed in the previous sections. It first consists of a precomputation phase where special messages and their corresponding error vectors are generated. Secondly, the messages are submitted for decryption and some decryption errors are observed. Finally, a phase with a statistical analysis of the messages/errors causing the decryption errors reveals the secret key.

In Section 7 we apply the model to to ss-ntru-pke, a version of NTRUEncrypt targeting the security level of NIST-V. The proposed attack is an adaptive CCA attack with complexity below the claimed security level. We provide a Rust implementation[2] where parts of the attack are simulated.

2 Preliminaries

2.1 Notation

Let $\mathbb{Z}_q$ be the ring of integers modulo q represented in $(-q/2, q/2]$, let R_q denote the ring $\mathbb{Z}_q[X]/(X^n + 1)$ and let $R_q^{k_1 \times k_2}$ denote the ring of $k_1 \times k_2$ matrices over R_q. Matrices will be represented with bold uppercase letters, while vectors are represented in bold lowercase. Let $\boldsymbol{A}_{ij}$ denote the element on the i^{th} row and j^{th} column of matrix $\boldsymbol{A}$, and let $\boldsymbol{A}_{ijk}$ denote the k^{th} coefficient of this element. Denote with $\boldsymbol{A}_{:j}$ the j^{th} column of $\boldsymbol{A}$.

[1] The software is available at `https://github.com/danversjp/failureattack`

[2] The software is available at `https://github.com/atneit/ss-ntru-pke-attack-simulation`

The rounding operation $\lfloor x \rceil_{q \to p}$ is defined as $\lfloor p/q \cdot x \rceil \in \mathbb{Z}_p$ for an element $x \in \mathbb{Z}_q$, while $\mathsf{abs}(\cdot)$ takes the absolute value of the input. These operations are extended coefficient-wise for elements of R_q and $R_q^{k_1 \times k_2}$. The two-norm of a polynomial $a \in R_q$ is written as $\|a\|_2$ and defined as $\sqrt{\sum_i a_i^2}$, which is extended to vectors as $\|\boldsymbol{a}\|_2 = \sqrt{\sum_i \|\boldsymbol{a}_i\|_2^2}$. The notation $a \leftarrow \chi(R_q)$ will be used to represent the sampling of $a \in R_q$ according to the distribution χ. This can be extended coefficient-wise for $\boldsymbol{A} \in R_q^{k_1 \times k_2}$ and is denoted as $\boldsymbol{A} \leftarrow \chi(R_q^{k_1 \times k_2})$. Let $\mathcal{U}$ be the uniform distribution. Denote with $\chi_1 * \chi_2$ the convolution of the two distributions χ_1 and χ_2, and denote with $\chi^{*n} = \underbrace{\chi * \chi * \chi * \cdots * \chi * \chi}_{n}$ the n^{th} convolutional power of χ.

2.2 Cryptographic definitions

A Public Key Encryption (PKE) is defined as a triple of functions PKE = (KeyGen, Enc, Dec), where the key generation KeyGen returns a secret key sk and a public key pk, where the encryption Enc produces a ciphertext c from the public key pk and the message $m \in \mathcal{M}$, and where the decryption Dec returns the message m' given the secret key sk and the ciphertext c.

A Key Encapsulation Mechanism (KEM) consists of a triple of functions KEM = ($\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps}$), where KeyGen generates the secret and public keys sk and pk respectively, where Encaps generates a key $k \in \mathcal{K}$ and a ciphertext c from a public key pk, and where Decaps requires the secret key sk, the public key pk and the ciphertext c to return a key k or the decryption failure symbol $\bot$. The security of a KEM can be defined under the notion of indistinguishability under chosen ciphertext attacks (IND-CCA).

$$\mathrm{Adv}^{\text{ind-cca}}_{\text{KEM}}(A) = \left| P\left(b' = b : \begin{array}{c} (pk, sk) \leftarrow \mathsf{KeyGen}(),\ b \leftarrow \mathcal{U}(\{0,1\}), \\ (c, d, k_0) \leftarrow \mathsf{Encaps}(pk), \\ k_1 \leftarrow \mathcal{K},\ b' \leftarrow A^{\mathsf{Decaps}}(pk, c, d, k_b), \end{array} \right) - \frac{1}{2} \right|$$

2.3 LWE/LWR problems

The decisional Learning with Errors problem (LWE) [Reg05] consists of distinguishing a uniform sample $(\boldsymbol{A}, \boldsymbol{U}) \leftarrow \mathcal{U}(\mathbb{Z}_q^{k_1 \times k_2} \times \mathbb{Z}_q^{k_1 \times m})$ from an LWE-sample $(\boldsymbol{A}, \boldsymbol{B} = \boldsymbol{AS} + \boldsymbol{E})$, were $\boldsymbol{A} \leftarrow \mathcal{U}(\mathbb{Z}_q^{k_1 \times k_2})$ and where the secret vectors $\boldsymbol{S}$ and $\boldsymbol{E}$ are generated from the small distributions $\chi_s(\mathbb{Z}_q^{k_2 \times m})$ and $\chi_e(\mathbb{Z}_q^{k_1 \times m})$ respectively. The search LWE problem states that it is hard to recover the secret $\boldsymbol{S}$ from the LWE sample.

This definition can be extended to Ring- or Module-LWE [LPR10; LS15] by using vectors of polynomials. In this case, the problem is to distinguish the uniform sample $(\boldsymbol{A},\boldsymbol{U}) \leftarrow \mathcal{U}(R_q^{k_1\times k_2} \times R_q^{k_1\times m})$ from a generalized LWE sample $(\boldsymbol{A},\boldsymbol{B} = \boldsymbol{AS}+\boldsymbol{E})$ in which $\boldsymbol{A} \leftarrow \mathcal{U}(R_q^{k_1\times k_2})$ and where the secret vectors $\boldsymbol{S}$ and $\boldsymbol{E}$ are generated from the small distribution $\chi_s(R_q^{k_2\times m})$ and $\chi_e(R_q^{k_1\times m})$ respectively. Analogous to the LWE case, the search problem is to recover the secret $\boldsymbol{S}$ from a generalized LWE sample.

The decisional generalized Learning with Rounding (LWR) problem [BPR12] is defined as distinguishing the uniform sample $(\boldsymbol{A}, \lfloor \boldsymbol{U} \rceil_{q\to p})$, with $\boldsymbol{A} \leftarrow \mathcal{U}(R_q^{k_1\times k_2})$ and $\boldsymbol{U} \leftarrow \mathcal{U}(R_q^{k_1\times m})$ from the generalized LWR sample $(\boldsymbol{A},\boldsymbol{B} = \lfloor \boldsymbol{AS} \rceil_{q\to p})$ with $\boldsymbol{A} \leftarrow \mathcal{U}(R_q^{k_1\times k_2})$ and $\boldsymbol{S} \leftarrow \chi_s(R_q^{k_2\times m})$. In the analogous search problem, one has to find $\boldsymbol{S}$ from a generalized LWR sample.

2.4 (Ring/Module) LWE based encryption

Let gen be a pseudorandom generator that expands seed$_{\boldsymbol{A}}$ into a uniformly random distributed matrix $\boldsymbol{A} \in R_q^{k\times k}$. Define enc as an encoding function that transforms a message m into a polynomial representation, and dec as the inverse decoding function. A general (Ring/Module) LWE based PKE, consisting of a key generation, an encryption and a decryption phase, can then be constructed as described in Algorithms 1 to 3 respectively. The randomness required for the generation of the secrets $\boldsymbol{S}'_B$, $\boldsymbol{E}'_B$ and $\boldsymbol{E}''_B$ during the encryption, is generated pseudorandomly from the uniformly distributed seed r that is given as an input.

Algorithm 1 PKE.KeyGen

```
Input:
Output: Public key $pk = (\boldsymbol{B}, seed_{\boldsymbol{A}})$, secret key $sk = \boldsymbol{S}_A$.
  $seed_{\boldsymbol{A}} \leftarrow \mathcal{U}\left(\{0,1\}^{256}\right)$
  $\boldsymbol{A} \leftarrow \mathsf{gen}(\mathsf{seed}_{\boldsymbol{A}}) \in R_q^{l\times l}$
  $\boldsymbol{S}_A \leftarrow \chi_s(R_q^{l\times m}), \boldsymbol{E}_A \leftarrow \chi_e(R_q^{l\times m})$
  $\boldsymbol{B} = \lfloor \boldsymbol{AS}_A + \boldsymbol{E}_A \rceil_{q\to p}$
```

Algorithm 2 PKE.Enc

```
Input: Public key $pk = (\boldsymbol{B}, seed_{\boldsymbol{A}})$, message $m$, randomness $r$
Output: Ciphertext $c = (\boldsymbol{V}', \boldsymbol{B}')$
  $\boldsymbol{A} \leftarrow \mathsf{gen}(\mathsf{seed}_{\boldsymbol{A}}) \in R_q^{l\times l}$
  $\boldsymbol{S}'_B \leftarrow \chi_s(R_q^{l\times m}), \boldsymbol{E}'_B \leftarrow \chi_e(R_q^{l\times m})$
  $\boldsymbol{E}''_B \leftarrow \chi_e(R_q^{m\times m})$
  $\boldsymbol{B}_r = \lceil \boldsymbol{B} \rfloor_{p\to q}$
  $\boldsymbol{B}' = \lfloor \boldsymbol{A}^T \boldsymbol{S}'_B + \boldsymbol{E}'_B \rceil_{q\to p}$
  $\boldsymbol{V}' = \lfloor \boldsymbol{B}_r^T \boldsymbol{S}'_B + \boldsymbol{E}''_B + \mathsf{enc}(m) \rceil_{q\to t}$
```

Algorithm 3 PKE.Dec

```
Input: Secret key $sk = \boldsymbol{S}_A$, ciphertext $c = (\boldsymbol{V}', \boldsymbol{B}')$
Output: Message $m'$
  $\boldsymbol{B}'_r = \lfloor \boldsymbol{B}' \rceil_{p \to q}$
  $\boldsymbol{V}'_r = \lfloor \boldsymbol{V}' \rceil_{t \to q}$
  $\boldsymbol{V} = \boldsymbol{B}'^T_r \boldsymbol{S}_A$
  $m' = \mathsf{dec}(\boldsymbol{V}'_r - \boldsymbol{V})$
```

Using this general framework, specific schemes can be described with appropriate parameter choices. When the ring R_q is chosen as $\mathbb{Z}_q$, the encryption is LWE-based as can be seen in FrodoKEM [Nae+17] and Emblem [Seo+17]. A value of $l = 1$ indicates a Ring-LWE based scheme including New Hope [Alk+16], LAC [Lu+17], LIMA [Sma+17] or R.Emblem [Seo+17]. If $l \neq 1$ and $R_q \neq \mathbb{Z}_q$, the scheme is based on the Module-LWE hard problem such as Kyber [Bos+17]. When referring to Kyber throughout this paper, we will consider the original version that includes rounding. The special case that $\chi_e = 0$ corresponds to (Module/Ring)-LWR-based schemes such as Round2 [Baa+17] and Saber [DAn+18]. In Lizard [Che+16], a combination of an LWE and LWR problem is proposed. In most (Ring/Module) LWE based schemes, $q = p$ and no rounding is performed in the calculation of $\boldsymbol{B}$ and $\boldsymbol{B}'$, while t is in most schemes much smaller than q leading to a drastic rounding of $\boldsymbol{V}'$.

We define $\boldsymbol{U}_A$, $\boldsymbol{U}'_B$ en $\boldsymbol{U}''_B$ as the errors introduced by the rounding operations, which is formalized as follows:

$$\boldsymbol{U}_A = \boldsymbol{A}\boldsymbol{S}_A + \boldsymbol{E}_A - \boldsymbol{B}_r \,, \tag{1}$$

$$\boldsymbol{U}'_B = \boldsymbol{A}^T\boldsymbol{S}'_B + \boldsymbol{E}'_B - \boldsymbol{B}'_r \,, \tag{2}$$

$$\boldsymbol{U}''_B = \boldsymbol{B}^T_r\boldsymbol{S}'_B + \boldsymbol{E}''_B + \mathsf{enc}(m) - \boldsymbol{V}'_r \,. \tag{3}$$

Let $\boldsymbol{S}$ be the vector constructed as the concatenation of the vectors $-\boldsymbol{S}_A$ and $\boldsymbol{E}_A + \boldsymbol{U}_A$, let $\boldsymbol{C}$ be the concatenation of $\boldsymbol{E}'_B + \boldsymbol{U}'_B$ and $\boldsymbol{S}'_B$, and let $\boldsymbol{G} = \boldsymbol{E}''_B + \boldsymbol{U}''_B$. An attacker that generates ciphertexts can compute $\boldsymbol{C}$ and $\boldsymbol{G}$ and tries to obtain information about $\boldsymbol{S}$. These variables are summarized below:

$$\boldsymbol{S} = \begin{pmatrix} -\boldsymbol{S}_A \\ \boldsymbol{E}_A + \boldsymbol{U}_A \end{pmatrix}, \quad \boldsymbol{C} = \begin{pmatrix} \boldsymbol{E}'_B + \boldsymbol{U}'_B \\ \boldsymbol{S}'_B \end{pmatrix}, \quad \boldsymbol{G} = \boldsymbol{E}''_B + \boldsymbol{U}''_B \,. \tag{4}$$

After the execution of this protocol, the two parties will arrive at the same key if the decoding $\mathsf{dec}(\boldsymbol{V}'_r - \boldsymbol{V})$ equals m. The term $\boldsymbol{V}'_r - \boldsymbol{V}$ can be rewritten as $(\boldsymbol{E}_A + \boldsymbol{U}_A)^T\boldsymbol{S}'_B - \boldsymbol{S}^T_A(\boldsymbol{E}'_B + \boldsymbol{U}'_B) + (\boldsymbol{E}'' + \boldsymbol{U}''_B) + \mathsf{enc}(m) = \boldsymbol{S}^T\boldsymbol{C} + \boldsymbol{G} + \mathsf{enc}(m)$.

The message can be recovered if and only if $\mathsf{abs}(\boldsymbol{S}^T\boldsymbol{C} + \boldsymbol{G}) < q_t$ for a certain threshold q_t that is scheme dependent.

We will say that a (decryption) failure occurred if the parties do not arrive at a common key due to a coefficient of $\mathsf{abs}(\boldsymbol{S}^T\boldsymbol{C}+\boldsymbol{G})$ that is larger than q_t, and will define $F(\boldsymbol{C}, \boldsymbol{G})$ as the probability of a decryption failure given $\boldsymbol{C}$ and $\boldsymbol{G}$ averaged over all $\boldsymbol{S}$, which can be expressed as $\sum_{\boldsymbol{S}} P(\mathsf{abs}(\boldsymbol{S}^T\boldsymbol{C} + \boldsymbol{G}) > q_t \mid \boldsymbol{S})P(\boldsymbol{S})$.

2.5 Fujisaki-Okamoto transformation

Using the Fujisaki-Okamoto transform [FO99; HHK17], one can transform a chosen plaintext secure PKE to an IND-CCA secure KEM. On top of the encryption from the PKE, the KEM defines an encapsulation and decapsulation function as described in Algorithms 4 and 5, using hash functions $\mathcal{H}$ and $\mathcal{G}$.

Algorithm 4 KEM.Encaps

```
Input: Public key $pk$
Output: Ciphertext $c$, key $K$
$m \leftarrow \mathcal{U}(\{0,1\}^{256})$
$r = \mathcal{G}(m)$
$c = \mathsf{PKE.Enc}(pk, m, r)$
$K = \mathcal{H}(r)$
```

Algorithm 5 KEM.Decaps

```
Input: Public key $pk$, secret key $sk$, ciphertext $c$
Output: Key $K$ or $\perp$
$m' = \mathsf{PKE.Dec}(sk, c)$
$r' = \mathcal{G}(m')$
$c' = \mathsf{PKE.Enc}(pk, m', r')$
if $c = c'$ then
    $K = \mathcal{H}(r)$
else
    $K = \perp$
end if
```

3 Weak-ciphertext failure boosting

In this section, we will develop a method to estimate the minimum amount of work to obtain one ciphertext that triggers a decryption failure. In contrast to an honest party that generates ciphertexts randomly, an attacker can search for ciphertexts that have a higher failure probability than average, which will be called 'weak'. As $\boldsymbol{C}$ and $\boldsymbol{G}$ are the only terms with which an attacker can influence decryption failures, the search for weak ciphertexts boils down to the search for weak $(\boldsymbol{C}, \boldsymbol{G})$. However, the pair $(\boldsymbol{C}, \boldsymbol{G})$ is generated through a hash $H()$ with random seed r, and during decryption it is checked whether the generator of the ciphertext knew the preimage r of $(\boldsymbol{C}, \boldsymbol{G})$. Therefore an attacker is forced to resort to a brute force search, which can be sped up at most quadratically using Grover's algorithm [Gro96].

To find a criterion for our search, we sort all possible $(\boldsymbol{C}, \boldsymbol{G})$ according to an increasing failure probability $F(\boldsymbol{C}, \boldsymbol{G})$. This list can then be divided into two sets using a threshold failure probability f_t: weak vectors with a failure probability higher or equal than f_t, and strong vectors with lower failure probability. Let $f()$ be the deterministic function that generates $\boldsymbol{C}$ and $\mathbf{G}$ from the random seed r. For a certain f_t, we can calculate the probability of generating a weak pair: $\alpha = P(F(\boldsymbol{C}, \boldsymbol{G}) > f_t \,|\, r \leftarrow \mathcal{U}, (\boldsymbol{C}, \boldsymbol{G}) = f(H(r)))$, and the probability of a decryption failure when a weak pair is used: $\beta = P(\mathsf{abs}(\boldsymbol{S}^T\boldsymbol{C} + \boldsymbol{G}) > q_t \,|\, r \leftarrow \mathcal{U}, (\boldsymbol{C}, \boldsymbol{G}) = f(H(r)), F(\boldsymbol{C}, \boldsymbol{G}) > f_t)$.

The amount of work for an adversary to find a weak pair $(\boldsymbol{C}, \boldsymbol{G})$ is proportional to α^{-1}, but can be sped up quadratically using Grover's algorithm on a quantum computer, resulting in an expected workload of $\sqrt{\alpha^{-1}}$. On the other hand, the probability of a decryption failure given a weak pair cannot be improved using quantum computation assuming that the adversary has no quantum access to the decryption oracle. This assumption is in agreement with the premise in the NIST Post-Quantum Standardization Call for Proposals [NIS16]. The expected work required to find a decryption failure given f_t is therefore the expected number of queries using weak ciphertexts times the expected amount of work to find a weak ciphertext, or $(\alpha \cdot \beta)^{-1}$ with a classical computer and $(\sqrt{\alpha} \cdot \beta)^{-1}$ with a quantum computer. An optimization over f_t gives the minimal effort to find one decryption failure.

3.1 Practical calculation

For most schemes, the full sorted list $(\boldsymbol{C}, \boldsymbol{G})$ is not practically computable, but using some observations and assumptions, an estimate can be found. The next three steps aim at calculating the distribution of the failure probability $F(\boldsymbol{C}, \boldsymbol{G})$, i.e. what is the probability of finding a $(\boldsymbol{C}, \boldsymbol{G})$ pair with a certain failure probability f. This distribution gives enough information to calculate α and β for a certain f_t.

First, we can remove the hash $H(.)$ in the probability expression by assuming the output of $f(H(.))$ given random input r to behave as the probability distributions (χ_C, χ_G), resulting in: $\alpha = P(F(\boldsymbol{C}, \boldsymbol{G}) > f_t \,|\, (\boldsymbol{C}, \boldsymbol{G}) \leftarrow (\chi_C, \chi_G))$ and $\beta = P(\mathsf{abs}(\boldsymbol{S}^T\boldsymbol{C} + \boldsymbol{G}) > q_t \,|\, (\boldsymbol{C}, \boldsymbol{G}) \leftarrow (\chi_C, \chi_G), F(\boldsymbol{C}, \boldsymbol{G}) > f_t)$.

Secondly, we assume that the coefficients of $\boldsymbol{S}^T\boldsymbol{C}$ are normally distributed, which is reasonable as the coefficients are a sum of $2(l \cdot n)$ distributions that are somewhat close to a Gaussian. The coefficients of the polynomial $(\boldsymbol{S}^T\boldsymbol{C})_{ij}$ will be distributed with mean $\mu = 0$ because of symmetry around 0, while the variance can be calculated as follows, after defining χ_{e+u} as the distribution of the coefficients of $\boldsymbol{E}_A + \boldsymbol{U}_A$:

$$\begin{aligned}
\mathsf{var}((\boldsymbol{S}^T\boldsymbol{C})_{ijk}) =& \mathsf{var}(\sum_{i=0}^{l-1}\sum_{k=0}^{n-1}\boldsymbol{C}_{ijk}s_{ijk} + \sum_{i=l}^{2l-1}\sum_{k=0}^{n-1}\boldsymbol{C}_{ijk}e_{ijk}) && (5)\\
&\text{where: } s_{ijk} \leftarrow \chi_s \text{ and } e_{ijk} \leftarrow \chi_{e+u} && (6)\\
=& \sum_{i=0}^{l-1}\sum_{k=0}^{n-1}\boldsymbol{C}_{ijk}^2\mathsf{var}(\chi_s) + \sum_{i=l}^{2l-1}\sum_{k=0}^{n-1}\boldsymbol{C}_{ijk}^2\mathsf{var}(\chi_{e+u}) && (7)\\
=& \|(\boldsymbol{E}'_B + \boldsymbol{U}'_B)_{:j}\|_2^2\mathsf{var}(\chi_s) + \|(\boldsymbol{S}'_B)_{:j}\|_2^2\mathsf{var}(\chi_{e+u})\,. && (8)
\end{aligned}$$

Therefore, the variance of the coefficients of $\boldsymbol{S}^T\boldsymbol{C}$ for a given $\boldsymbol{C}$ is the same for all coefficients in the same column. This variance will be denoted as σ_j^2 for coefficients in the j^{th} column of $\boldsymbol{S}^T\boldsymbol{C}$. Furthermore, following the Gaussian assumption, the failure probability given σ_j^2 is the same as the failure probability given the j^{th} column of $\boldsymbol{C}$.

In the third step we gradually calculate the distribution of the failure probability. We start from the distribution of the failure probability of the coefficient at the ijk^{th} position given σ_j, denoted with $\chi_{coef\,|\,\sigma}$. This distribution expresses the probability of finding a $\boldsymbol{G}$ so that the failure probability is equal to f_{ijk} given a certain value of $\boldsymbol{C}$ (or equivalently σ_j^2) and can be expressed as follows:

$$P(f_{ijk}\,|\,\boldsymbol{G} \leftarrow \chi_G, \boldsymbol{C})\,, \tag{9}$$

$$\tag{10}$$

where:

$$\begin{aligned}
f_{ijk} &= P(\mathsf{abs}(\boldsymbol{S}^T\boldsymbol{C} + \boldsymbol{G})_{ijk} > q_t\,|\,\boldsymbol{G}, \boldsymbol{C}) && (11)\\
&\approx P(\mathsf{abs}(x + \boldsymbol{G}_{ijk}) > q_t\,|\,\boldsymbol{G}, x \leftarrow \mathcal{N}(0, \sigma_j^2), \sigma_j^2)\,. && (12)
\end{aligned}$$

The distribution $\chi_{col\,|\,\sigma}$, which models the probability of a failure in the j^{th} column of $\mathsf{abs}(\boldsymbol{S}^T\boldsymbol{C} + \boldsymbol{G})$ given σ_j^2, can be calculated using the convolution of the distributions of the mn individual coefficient failures $\chi_{coef\,|\,\sigma}$ as follows:

$$\chi_{col\,|\,\sigma} = \chi_{coef\,|\,\sigma}^{*nm}\,. \tag{13}$$

The conditioning on σ_j^2 is necessary to counter the dependency between the coefficients of the columns of $\mathsf{abs}(\boldsymbol{S}^T\boldsymbol{C}+\boldsymbol{G})$, which are dependent as a result of sharing the same variance σ_j^2.

The distribution of failure probabilities in the j^{th} column of $\boldsymbol{S}^T\boldsymbol{C}$, denoted as χ_{col}, can then be calculated using a weighted average over the possible values of σ_j^2 as follows:

$$\chi_{col} = \sum_{l_c} P(f \mid f \leftarrow \chi_{col,\sigma}^{*nm}) P(\sigma_j^2 = l_c)\,. \tag{14}$$

Finally we can calculate the full failure distribution χ_{FAIL} as the convolution of the m probability distributions corresponding to the failure distributions of the different columns. This convolution does not have the dependency on σ_j^2 as failures of different columns are independent conditioned on $\boldsymbol{C}$ and $\boldsymbol{G}$, therefore:

$$\chi_{\text{FAIL}} = \chi_{col}^{*m}\,. \tag{15}$$

From this failure distribution, we can calculate α and β for an arbitrary value of f_t:

$$\alpha = P(f > f_t \mid f \leftarrow \chi_{\text{FAIL}})\,, \tag{16}$$

$$\beta = \frac{\sum_{f>f_t} f \cdot P(f \mid f \leftarrow \chi_{\text{FAIL}})}{\alpha}\,. \tag{17}$$

We want to stress that this calculation is not exact, mainly due to the Gaussian assumption in the second step. More accurate estimates could be obtained with a more accurate approximation in step 2, tailored for a specific scheme. In this case, the assumptions and calculations of step 1 and step 3 remain valid. For the estimations of LAC [Lu+17] in subsequent paragraphs, we followed their approach for the calculation of the effect of the error correcting code. Note that this is not an exact formula as the inputs of the error correcting code are correlated through their polynomial structure.

In Fig. 1 we compare the values of α and β calculated using the technique described above, with exhaustively tested values on a variant of LAC128 without error correction. For step 2 of the practical calculation, we use both a Gaussian approximation as well as a binomial approximation that is more tailored for LAC. We can observe that our estimation of the effect of failure boosting is relatively close to reality.

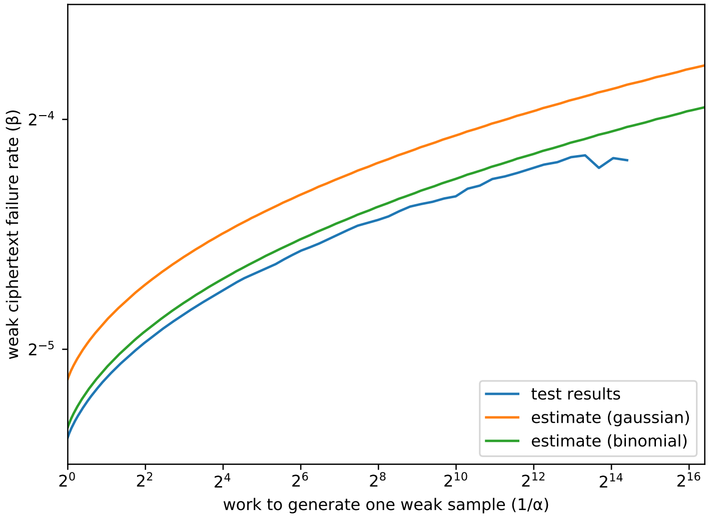


Figure 1: The failure rate of one weak ciphertext (β) as a function of the work required to generate one weak ciphertext (α) on a classical computer for LAC128 without error correction.

3.2 Applications of failure boosting

Failure boosting is a useful technique in at least three scenarios: first, if there is no multi-target protection, second, if the adversary can only perform a limited number of queries to the decryption oracle and third, if the adversary has access to a quantum computer.

In some (Ring/Module) LWE/LWR schemes, the seed r is the only input to the pseudorandom generator that generates $\boldsymbol{C}$ and $\boldsymbol{G}$. This paves the way to a multi-target attack where precomputed weak values of r can be used against multiple targets: after choosing the parameter f_t, the adversary can generate weak ciphertexts in approximately α^{-1} time ($\sqrt{\alpha^{-1}}$ if he has access to a quantum computer). Each precomputed sample has then a failure probability of β against every target. Fig. 2 shows the failure probability of one weak ciphertext versus the amount of work to generate that ciphertext on a classical computer. Multi-target protection, for example by including the public key into the generation of $\boldsymbol{C}$ en $\boldsymbol{G}$ as proposed in Kyber [Bos+17] and Saber [DAn+18] is a relatively cheap option to resolve this issue.

If the adversary can only perform a limited number of decryption queries, for example 2^{64} in the NIST Post-Quantum Standardization Call for Proposals [NIS16], the adversary can use failure boosting to reduce the number of required decryption queries. To this end, he chooses the parameter f_t so that the inverse of the failure probability β^{-1} equals the decryption query limit n_d, which

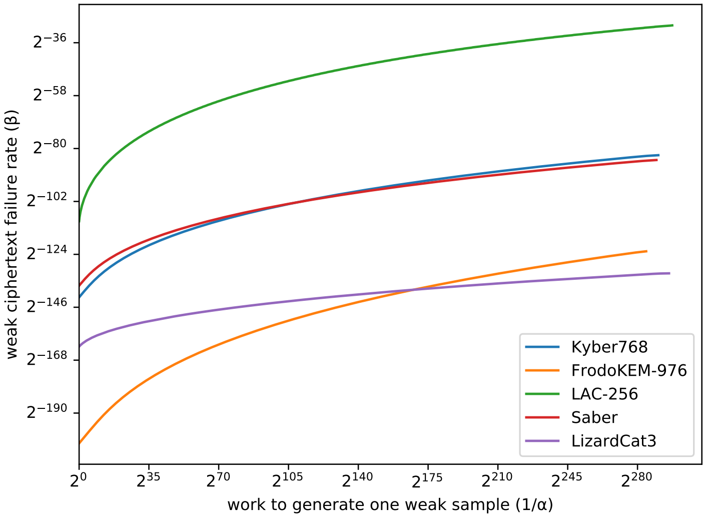


Figure 2: The failure rate of one weak ciphertext (β) as a function of the work required to generate one weak ciphertext (α) on a classical computer.

results in a probability of finding a decryption failure of approximately $(1-e^{-1}) \approx 0.63$. To find i failures with similar probability, the failure probability should be brought up so that $\beta^{-1} = n_d/i$. Since the amount of work to generate one input of the decryption query is approximately α^{-1} ($\sqrt{\alpha^{-1}}$ quantumly), the total amount of work expected is $\alpha^{-1}\beta^{-1}$, ($\sqrt{\alpha^{-1}}\beta^{-1}$ quantumly). Fig. 3 shows the expected total amount of work to find one decryption failure with a classical computer, versus the failure rate of one weak ciphertext.

An adversary with a quantum computer always benefits from failure boosting, as the search for weak ciphertexts can be sped up using Grover's algorithm. However, this speedup is not quadratic if the adversary has no quantum access to the decryption oracle. Fig. 4 shows the total amount of expected work to find one decryption failure, versus the amount of work to find one weak ciphertext on a quantum computer $\sqrt{\alpha^{-1}}$.

4 Estimation of the secret

Finding a decryption failure does not immediately break the security of the KEM, but it does provide extra information to an adversary. In this section we will investigate how much this information leaks about the secret. An adversary that has obtained ciphertexts that produce decryption failures can use them to make an estimation of the secret $\boldsymbol{S}$.

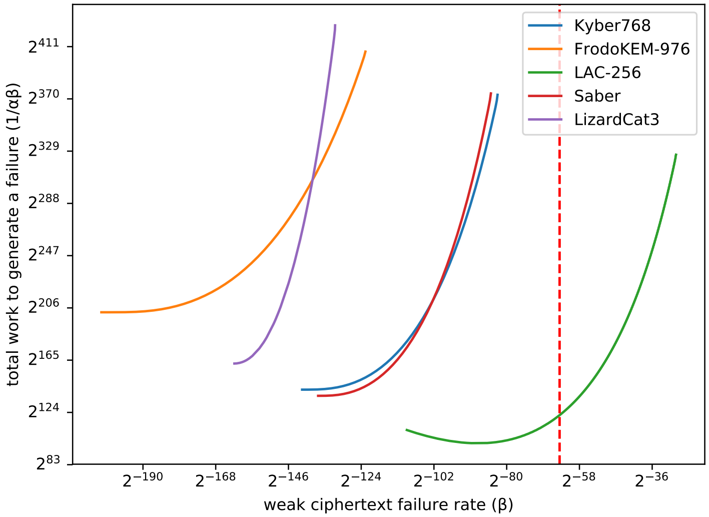


Figure 3: The expected amount of work ($\alpha^{-1}\beta^{-1}$) on a classical computer, as a function of the failure rate of one weak ciphertext (β). The red dotted line indicates a failure rate of 2^{-64}.

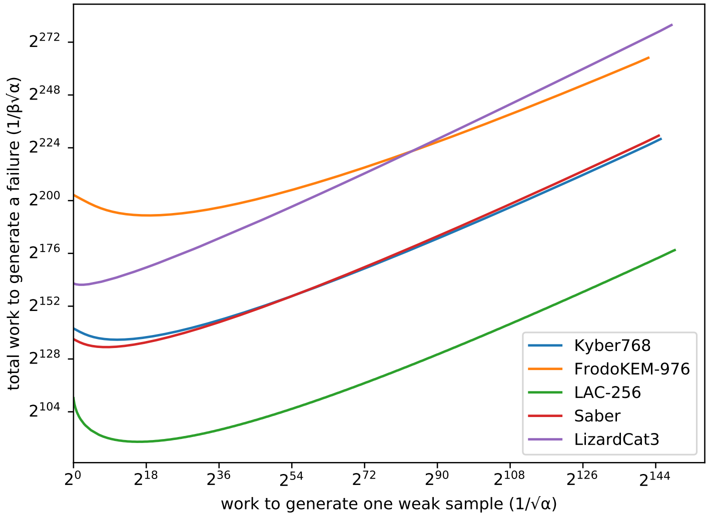


Figure 4: The expected amount of work ($\sqrt{\alpha^{-1}}\beta^{-1}$) as a function of the work required to generate one weak ciphertext ($\sqrt{\alpha^{-1}}$) on a classical computer.

When a failure occurs, we know that at least one coefficient of $\mathsf{abs}(\boldsymbol{S}^T\boldsymbol{C}+\boldsymbol{G})$ is larger than the threshold q_t. This leads to a classification of the coefficients in the set of fail coefficients v_f and no-fail coefficients v_s. To each coefficient at position (i, j, k), a vector of integers $\boldsymbol{s}$ can be associated by taking the coefficients of $\boldsymbol{S}_{:i}$. Similarly, the coefficient can be linked to a vector of integers $\boldsymbol{c}$ calculated as a function of $\boldsymbol{C}_{:j}$ and k, so that the multiplication $\boldsymbol{sc}$ equals that coefficient.

No-fail vectors will contain negligible information about the secret $\boldsymbol{s}$, but failure vectors do carry clues, as the threshold exceeding value of the coefficients of $\boldsymbol{S}^T\boldsymbol{C} + \boldsymbol{G}$ implies a correlation between the corresponding $\boldsymbol{c}$ and $\boldsymbol{s}$. This correlation can be positive, in case of a large positive value of the coefficient, or negative, in case of a large negative value of the coefficient. Consequently, the fail coefficients can be further divided into the sets of positive v_{fp} and negative v_{fn} fail coefficients respectively. Moreover, negative fail vectors can be transformed into positive fail vectors by multiplication with -1. Note that failure coefficients at position (i, j, k) will only contain information about the j^{th} column of $\boldsymbol{S}$, which is why the estimation of the columns of $\boldsymbol{S}$ can be performed independently.

4.1 One positive failure vector

We will first examine the case where we know one positive fail vector $\boldsymbol{c}$ and associated coefficient $\boldsymbol{G}_{i,j,k}$, which we will denote with g. This corresponds to the case where one failing ciphertext and the location and sign of the error is known. The question is how much the knowledge about $\boldsymbol{c}$ and g can improve our estimate of the associated secret $\boldsymbol{s}$. Applying Bayes' theorem and assuming independence between the coefficients of $\boldsymbol{c}$ and $\boldsymbol{s}$ that are on different positions, we can write:

$$\begin{aligned} P(\boldsymbol{s}_i \,|\, \boldsymbol{c}, g, \boldsymbol{sc} > q_t - g) \approx& P(\boldsymbol{s}_i \,|\, \boldsymbol{c}_i, g, \boldsymbol{sc} > q_t - g) && (18)\\ =& \frac{P(\boldsymbol{sc} > q_t - g \,|\, \boldsymbol{s}_i, \boldsymbol{c}_i, g) P(\boldsymbol{s}_i \,|\, \boldsymbol{c}_i, g)}{P(\boldsymbol{sc} > q_t - g \,|\, \boldsymbol{c}_i, g)} && (19)\\ =& \frac{P(\sum_j^{j \neq i} \boldsymbol{s}_j\boldsymbol{c}_j > q_t - g - \boldsymbol{s}_i\boldsymbol{c}_i \,|\, \boldsymbol{s}_i, \boldsymbol{c}_i, g) P(\boldsymbol{s}_i)}{P(\boldsymbol{sc} > q_t - g \,|\, \boldsymbol{c}_i, g)}\,. && (20) \end{aligned}$$

The improved estimates for the coefficients of $\boldsymbol{s}$ can in turn be used to get an estimate $\boldsymbol{s}_{est}$ that minimizes its variance $E[(\boldsymbol{s}_{est} - \boldsymbol{s})^2]$ as follows:

$$0 = \frac{d}{d\boldsymbol{s}_{est,i}} E((\boldsymbol{s}_{est,i} - \boldsymbol{s}_i)^2) \tag{21}$$

$$= 2 \sum_{\boldsymbol{s}_i} (\boldsymbol{s}_{est,i} - \boldsymbol{s}_i) P(\boldsymbol{s}_i)\,, \tag{22}$$

$$\text{or:} \quad \boldsymbol{s}_{est,i} = \sum_{\boldsymbol{s}_i} \boldsymbol{s}_i \cdot P(\boldsymbol{s}_i)\,. \tag{23}$$

The estimate of $\boldsymbol{s}$ gives the estimate of the j^{th} column of $\boldsymbol{S}$, which can be divided trivially in an approximation $\boldsymbol{S}_{A,est}$ of $(\boldsymbol{S}_A)_{:j}$ and $\boldsymbol{E}_{A,est}$ of $(\boldsymbol{E}_A + \boldsymbol{U}_A)_{:j}$. These vectors can be used to transform the original (Ring/Module) LWE/LWR sample $(\boldsymbol{A}, \boldsymbol{A}(\boldsymbol{S}_A)_{:j} + (\boldsymbol{E}_A + \boldsymbol{U}_A)_{:j})$ into a (Ring/Module) LWE alike problem with a smaller secret variance by subtracting $\boldsymbol{A}\boldsymbol{S}_{A,est} + \boldsymbol{E}_{A,est}$. This results in the sample $(\boldsymbol{A}, \boldsymbol{A}((\boldsymbol{S}_A)_{:j} - \boldsymbol{S}_{A,est}) + (\boldsymbol{E}_A + \boldsymbol{U}_A)_{:j} - \boldsymbol{E}_{A,est})$, which is a problem with smaller secret $(\boldsymbol{S}_A)_{:j} - \boldsymbol{S}_{A,est}$ and noise $(\boldsymbol{E}_A + \boldsymbol{U}_A)_{:j} - \boldsymbol{E}_{A,est}$. We will call this new problem the simplified problem.

4.2 Multiple fail vectors

Having access to m positive fail vectors $\boldsymbol{c}^{(1)} \dots \boldsymbol{c}^{(m)}$ from the same column, with corresponding values of G as $g^{(1)} \dots g^{(m)}$, an adversary can improve his estimate of $P(\boldsymbol{s})$ and therefore obtain a better estimate $\boldsymbol{s}_{est}$, assuming that the failure vectors $\boldsymbol{c}_i$ are independent conditioned on $\boldsymbol{s}$. This corresponds to knowing m failing ciphertexts and the location and sign of their errors.

$$P(\boldsymbol{s}_i \,|\, \boldsymbol{c}^{(1)} \dots \boldsymbol{c}^{(m)}, g^{(1)} \dots g^{(m)}) \approx P(\boldsymbol{s}_i \,|\, \boldsymbol{c}_i^{(1)} \dots \boldsymbol{c}_i^{(m)}, g^{(1)} \dots g^{(m)}) \tag{24}$$

$$= \frac{P(\boldsymbol{c}_i^{(1)} \dots \boldsymbol{c}_i^{(m)} \,|\, \boldsymbol{s}_i, g^{(1)} \dots g^{(m)}) P(\boldsymbol{s}_i \,|\, g^{(1)} \dots g^{(m)})}{P(\boldsymbol{c}_i^{(1)} \dots \boldsymbol{c}_i^{(m)} \,|\, g^{(1)} \dots g^{(m)})} \tag{25}$$

$$= \frac{P(\boldsymbol{s}_i) \prod_{k=1}^{m} P(\boldsymbol{c}_i^{(k)} \,|\, \boldsymbol{s}_i, g^{(k)})}{\prod_{k=1}^{m} P(\boldsymbol{c}_i^{(k)} \,|\, g^{(k)})}\,. \tag{26}$$

Similar to Eq. (20), $P(\boldsymbol{c}_i \,|\, \boldsymbol{s}_i, g^{(k)})$ can be calculated as:

$$P(\boldsymbol{c}_i \,|\, \boldsymbol{s}_i, g, \boldsymbol{sc} > q_t - g) = \frac{P(\boldsymbol{sc} > q_t - g \,|\, \boldsymbol{s}_i, \boldsymbol{c}_i, g) P(\boldsymbol{c}_i \,|\, \boldsymbol{s}_i, g)}{P(\boldsymbol{sc} > q_t - g \,|\, \boldsymbol{s}_i, g)} \tag{27}$$

$$= \frac{P(\sum_j^{j \neq i} \boldsymbol{s}_j \boldsymbol{c}_j > q_t - g - \boldsymbol{s}_i \boldsymbol{c}_i \,|\, \boldsymbol{s}_i, \boldsymbol{c}_i, g) P(\boldsymbol{c}_i)}{P(\boldsymbol{sc} > q_t - g \,|\, \boldsymbol{s}_i, g)}\,. \tag{28}$$

In subsequent calculations, each value of the coefficient of g is taken as the maximum possible value.

4.3 Classification of vectors

The above approach assumes a prior knowledge of the exact position and sign of the errors. This information is needed to link coefficients of $\boldsymbol{C}$ with their corresponding coefficient of $\boldsymbol{S}$. However, this is not always a trivial problem. For most schemes there are three sources of extra information that will allow to perform this classification with a high probability using only a few decryption failures.

Firstly, a large coefficient of $\boldsymbol{G}$ would induce a higher failure probability for the corresponding coefficient of the error term $\boldsymbol{S}^T\boldsymbol{C}+\boldsymbol{G}$. Thus, failures are more likely to happen at positions linked to that coefficient of $\boldsymbol{G}$. Moreover, a positive value of the coefficient suggests a positive error so that $\boldsymbol{c} \in v_{fp}$, while a negative value hints at a negative error, or $\boldsymbol{c} \in v_{fn}$.

Secondly, as vectors $\boldsymbol{c} \in v_f$ are correlated with the secret $\boldsymbol{s}$, they are also correlated with each other. Therefore, vectors $\boldsymbol{c} \in v_f$ are more correlated between each other than a vector $\boldsymbol{c} \in v_f$ with a vector $\boldsymbol{c} \in v_s$. Moreover, a high positive correlation suggests that the vectors share the same class v_{fp} or v_{fn}, while a high negative correlation indicates that the vectors have a different classes. This allows for a clustering of the fail vectors using the higher than average correlation, under the condition that the correlation difference is high enough. This correlation difference is related to the failure rate: a low failure rate implies a higher correlation because only ciphertexts that are highly correlated with the secret lead to a failure rate in this case. For example, Fig. 5 shows an estimate of the correlations between vectors of the classes v_{fp} (pos), v_{fn} (neg) and v_s (nofail) in Kyber768. This approach does not work for schemes with strong error correcting codes (ECC) such as LAC, as the bit error rate before correction is relatively high for these types of algorithms, leading to a relatively low correlation between failure vectors.

In case of a ring/module structure of the coefficients of $\boldsymbol{S}$, an additional structure arises leading to an artifact in which some pairs of no-fail coefficients within the same polynomial also have high correlation of their corresponding vectors. Imagine a pair of failure coefficients at positions (i, j, k_1) and (i, j, k_2) from different decryption failures a, b, with corresponding matrices $\boldsymbol{C}^{(a)}$ and $\boldsymbol{C}^{(b)}$. The correlation of the vectors $\boldsymbol{c}^{(a)}$ and $\boldsymbol{c}^{(b)}$ can be written as $X^{k_1}\boldsymbol{C}^{(a)T}_{:,j} X^{k_2}\boldsymbol{C}^{(b)}_{:,j} = X^{k_1+k_2}\boldsymbol{C}^{(a)T}_{:,j}\boldsymbol{C}^{(b)}_{:,j}$, from which is clear that the vectors from $\boldsymbol{C}^{(a)}$ and $\boldsymbol{C}^{(b)}$, with respective positions $(i, j, k_1 - t)$ and $(i, j, k_2 + t)$ have the same correlation. The clustering will thus result in n classes, with one class containing the failure vectors. Combining this information with the information of the first method gives an adversary the failure vectors with high probability. Otherwise, an adversary can estimate the secret n times and check the validity of the result using the (Ring/-Module) LWE/LWR problem.

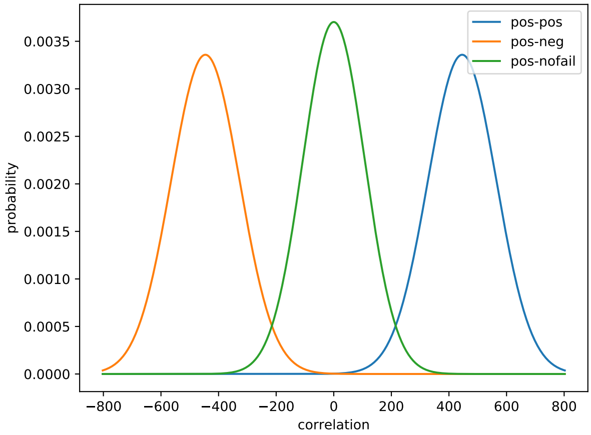


Figure 5: The probability of a certain value of the correlation between different classes of vectors in Kyber768.

Finally, for schemes that use error correcting codes to reduce their failure probability, side channel leakage during the error correction might reveal information on the presence or position of failure coefficients. Note that if this is the case, it might not even be necessary to obtain a decryption failure since failing coefficients could also be collected on successful decryptions where there is at least one failing coefficient.

4.4 Implications

Figure 6 depicts the relative variance reduction of the secret as a function of the number of positive failure vectors for various schemes. For schemes that have a very low failure probability for individual coefficients of $\boldsymbol{S}^T\boldsymbol{C}+\boldsymbol{G}$, such as Kyber, Saber and FrodoKEM, the variance of the secret drastically reduces upon knowing only a few failing ciphertexts. Assuming that the simplified problem, that takes into account the estimate of the secret, has the same complexity as a regular (Ring/-Module) LWE problem with similar secret variance, we can calculate the remaining hardness of the simplified problem as a function of the number of positive failure vectors as shown in Fig. 7 using the toolbox provided by Albrecht et al. [Alb+18] using the Q-core sieve estimate.

The effectiveness of the attack declines as the failure probability of the individual coefficients increases, since the correlation between the secret and the failing ciphertext is lower in this case. This can be seen in the case of LAC, where the individual coefficients have relatively high failure rates due to a strong error correcting code. On the other hand, a failing ciphertext will contain multiple errors, making it easier to collect multiple failure vectors.

Note that once one or more failures are found, they can be used to estimate the secret. This estimate in turn can be used to improve the search for weak ciphertexts by considering $F(\boldsymbol{C}, \boldsymbol{G})$ as $\sum_{\boldsymbol{S}} P(\text{FAIL}(\boldsymbol{C}, \boldsymbol{G}), \boldsymbol{S})$, where $\boldsymbol{S}$ is not sampled from $\chi_{\boldsymbol{S}}$, but from the new probability distribution $\chi_{\boldsymbol{S}_{est}}$. Therefore, the search for weak keys could become easier the more failures have been found. However, we do not take this effect into account in this paper.

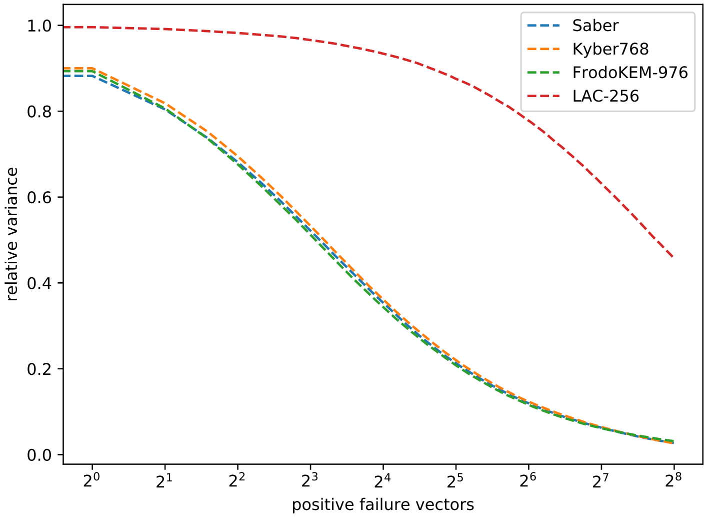


Figure 6: The relative reduction in entropy as a function of the number of positive failure vectors

5 Weak-ciphertext attack

Using the failure boosting technique from Section 3 and the secret estimation method from Section 4, we can lower the security of a (Ring/Module) LWE/LWR scheme on the condition that its failure rate is high enough. To this end, we first collect i decryption failures using the failure boosting technique, which would cost approximately $i\sqrt{\alpha^{-1}}\beta^{-1}$ work. Then, the exact error position and failure type should be determined for all of the failure vectors using the techniques of Section 4.3. Based on this information, the secret can be estimated, which in turn can be used to simplify the (Ring/Module) LWE/LWR problem. These last two operations require a negligible amount of work compared to finding decryption failures. Finally, we need to solve the simplified problem, with has a complexity $S_{\text{simplified}}(i)$ as estimated in Section 4. The total amount of work is therefore $\mathcal{O}(S_{\text{simplified}}(i) + i\sqrt{\alpha^{-1}}\beta^{-1})$, which is depicted in Fig. 8 as a function of the number of failures i. Note that the practical security of Kyber relies on an er-

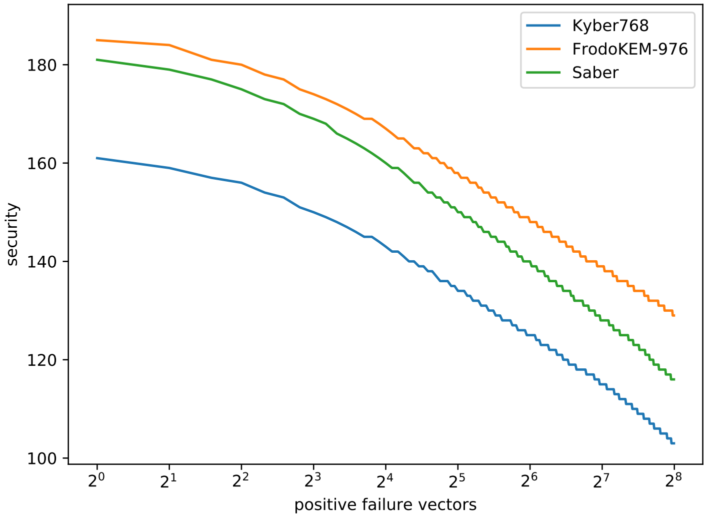


Figure 7: The hardness of the simplified problem as a function of the number of positive failure vectors

ror term $\boldsymbol{E}_A$ as well as a rounding term $\boldsymbol{U}_A$. Both are taken into account in the security calculation.

Table 1 gives an overview of the original hardness of the scheme before decryption failure usage S, and the attack cost $S_{\text{simplified}}(i) + i\sqrt{\alpha^{-1}}\beta^{-1}$ using decryption failures for ideal values of i and f_t, which are calculated through a brute force sweep. The number of collected decryption failures i and the expected number of decryption queries $i\beta^{-1}$ is also included. These values are calculated assuming that the adversary can perform an unlimited number of decryption queries. From this table we can see that the security of Kyber and Saber is considerably reduced. This is due to the fact that finding a failure is easier than breaking the security of the scheme S. For the case of FrodoKEM976, the security is not affected as the work to obtain a failure is considerably larger than breaking the security S.

In other situations such as a multi-target attack or having only a limited number of decryption queries, other values of f_t and i will obtain optimal results. For example in a multi-target attack scenario one would select a higher threshold f_t to be able to efficiently re-use the precomputation work α^{-1} for weak ciphertexts and therefore reduce the overall work. A limit on the number of decryptions n_d could make it necessary to increase the amount of precomputational work α^{-1} in order to reduce the failure rate $\beta^{-1} < n_d/i$. This would make the attack more expensive or might even invalidate it. For example, the NIST Post-Quantum Standardization Process decryption limit is set to 2^{64}, which rules out a decryption failure attack on schemes with a low enough failure rate such as Saber and Kyber, which

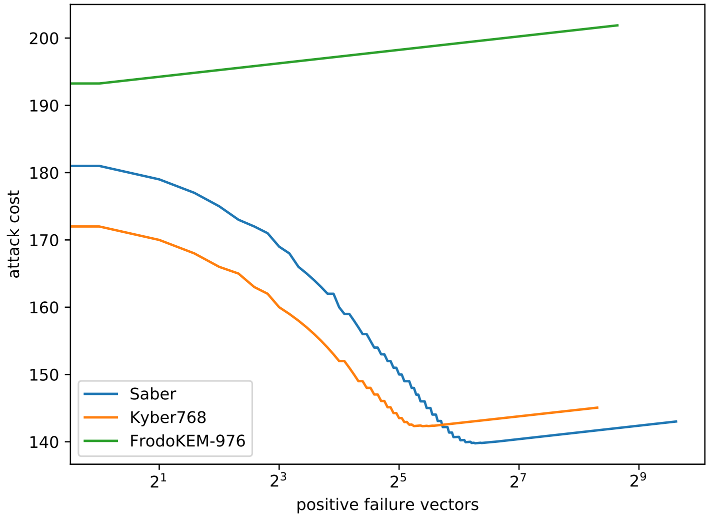


Figure 8: The full amount of work to break the scheme as a function of the number of collected decryption failures

can be deduced from Fig. 3. As such, the security of this schemes is not affected within the NIST framework.

6 A weak-key attack model

In this section we elaborate a weak-key (multi-target) attack model when the adversary can only have a limited number of decryption queries to one user but multiple users can be queried. We observe that for certain keys, the error probability can be much higher when applying the failure boosting technique, i.e., choosing 'weak' ciphertexts as discussed in Section 3, if the chosen ciphertexts satisfy certain key-related properties. The major targets are the same as before – lattice-based NIST post-quantum proposals with CCA security using some CCA transformations.

We set the maximum number of ciphertexts that can be submitted to a node with a public key to be 2^K and we set the maximum number of public keys in the system to be 2^L. Refering again to the NIST Post-Quantum Standardization Process, they have indicated in their call that at least $K = 64$ can be considered. In the discussion forum [NIS19] for the same project, we have also seen researchers mentioning that $L = 64$ can be considered. We will adopt $K = L = 64$ in the further sections since it seems these values are not questioned, although larger values of K and L can give more powerful attacks and could definitely be relevant. For example, comparing with attacks on symmetric schemes, such attacks

Table 1: The security of different schemes with and without decryption failures

	original security	attack cost	reduction factor	decryption failures	decryption queries
Saber	2^{184}	2^{139}	2^{45}	77	2^{131}
FireSaber	2^{257}	2^{170}	2^{87}	233	2^{161}
Kyber768	2^{175}	2^{142}	2^{33}	42	2^{131}
Kyber1024	2^{239}	2^{169}	2^{70}	159	2^{158}
LAC256	2^{293}	$2^{97\dagger}$	2^{196}	$106 \cdot 56$	2^{80}
FrodoKEM976	2^{188}	2^{188}	2^{0}	0	0

† Note that it seems not straightforward for LAC256 to obtain the exact position and type of the errors, which is required to obtain this result

may require a number of plaintext-ciphertext pairs that are close to the number of possible keys (like 2^{200}), and still they are considered valid attacks.

The proposed attack procedure is split in three steps.

1. Do a precomputation step to establish pairs of messages and corresponding ciphertexts and let informally the set $\mathcal{F}$ denote error vectors corresponding to the different messages, which are equivalent to the $(\boldsymbol{C}, \boldsymbol{G})$ pairs chosen before. These selected error vectors should be with particular properties, e.g, with large norm and/or with several large entries in certain positions, etc.

2. Send the ciphertexts contained in $\mathcal{F}$ and assume that we learn the decrypted messages. Assume further that a subset have been erroneously decrypted (wrong decoding due to too large error) and let $\mathcal{F}'$ be the error vectors causing decryption failure. The cardinality of this set could be larger than average if certain properties (related to $\mathcal{F}$) of the secret vector hold. So we submit the set of ciphertexts to each node holding a public key. The node giving the largest decryption failure rate is selected as the target public key for the attack.

3. Do statistical testing on the set $\mathcal{F}'$ (and possibly the set $\mathcal{F}$) to establish relationships between the secret key and given the noise vectors leading to a decryption failure. Analyzing their correlation, we may be able to recover partial secrets, which can considerably reduce the solving complexity of the underlying hard problem. We are then able to perform a full key-recovery attack via classic approaches such as lattice reduction algorithms.

Note that the above procedure is very close to the weak-ciphertext attack described in the previous sections. One major difference is that here we choose the set $\mathcal{F}$ of 'weak' ciphertexts to be related to the 'weak' keys targeted, while in the prior, the 'weak' ciphertexts are chosen to have a larger decryption failure rate averaged over all keys.

We discuss the three steps briefly. In the precomputation step, we can observe a first difference between different schemes. Most schemes include the public key in the generation of the noisy vectors (as input in the hash function generating the noise). This means that a constructed set $\mathcal{F}$ can only be used for a single public key and a new such set must be constructed for the next public key. For simplicity, we assume $|\mathcal{F}| = 2^K$ and note that the number of nodes with a public key is 2^L. If we set the computational complexity of precomputing a set $\mathcal{F}$ to be 2^λ, the overall complexity of this first step is $2^{\lambda+L}$. On the other hand, there are also schemes where error vectors are generated independent of the public key (e.g. LAC). In such a case the same set $\mathcal{F}$ can be used on all public keys and the complexity is only 2^λ. We could also use Grover's search algorithm to accelerate the precomputation step, as discussed in Section 3. However, since the pre-quantum and post-quantum security goals in the NIST Post-Quantum Standardization Process are different for a certain security level, this quantum acceleration may not help us to break the claimed security level of a submission.

For the second step, the idea is that among many public keys, there will be one where the corresponding secret values have a property that causes more decryption errors than on average. So to increase the decryption error probability to a reasonable and detectable level, we consider that a special property in the secret value is held with probability at least p', where $0 < p' < 1$. We then assume that $p' = 2^{-L}$, so we can expect that this special property in the secret value holds for one public key. As mentioned, with respect to the CCA security, NIST restricts to have at most 2^{64} decryption calls to each user (public key). So in order to distinguish a special property in the secret value corresponding to a public key, one needs to get the failure rate for this case to be larger than 2^{-64}.

Finally, in the statistical testing part, we have a set of error vectors that have caused decryption errors. There seems to be a plethora of methods that can be used to recover secret values. For instance, the strong maximum-likelihood approach has been discussed in Section 4 and heuristic approaches can also be applied. A general approach that we can adopt is to consider a smaller part of the secret vector under reconstruction, and select the most probable values for this part, based on the observed error vectors in $\mathcal{F}$. Then one combines such guesses for different parts and builds an approximation of a secret vector. A good approximation will mostly be sufficient as it can be used in lattice-basis reduction algorithms.

We note that in many applications, the challenge is to detect the first decryption failure, since we can usually have adaptive approaches to find more failures afterwards with a lower complexity. This idea is further demonstrated in the next section where an adaptive CCA attack on ss-ntru-pke will be presented, and also in a code-based application [NJS19].

7 A weak-key attack on ss-ntru-pke

We have applied the described weak-key approach and provide the details of attacking ss-ntru-pke, a version in the submission to the NIST Post-Quantum Standardization Process – NTRUEncrypt [Zha+17]. Connected is also the provably secure NTRU [SS11] whose security is based purely on the hardness of Ring-LWE. NTRUEncrypt with different parameter choices has been around for a long time and is one of the most competitive lattice-based schemes when it comes to performance.

Note that our attack in this section is in the pre-quantum (classic) security framework due to the different security goal for NIST-V when Grover's algorithm is considered. We adopt the notations from the NTRUEncrypt submission [Zha+17] throughout this section.

7.1 The ss-ntru-pke scheme

ss-ntru-pke is the version of NTRUEncrypt targeting the highest security level, being 256 bits. This scheme achieves CCA2 security via the NAEP transform [How+03b], a transform similar to the Fujisaki-Okamoto transformation with an additional mask. We give a very brief explanation of the scheme. For most of the description and details, we refer to [Zha+17]. In the key generation (see Algorithm 6), two secret polynomials $\mathbf{f}, \mathbf{g} \in \mathcal{R}$ are selected, where the coordinates are chosen from a discrete Gaussian $\mathcal{X}_\sigma$ distribution with standard deviation σ. A public key is formed by computing $\mathbf{h} = \mathbf{g}/(p\mathbf{f} + 1)$.

Algorithm 6 ss-ntru-pke.KeyGen

```
Input: Parameter sets PARAM = $\{N, p, q, \sigma\}$ and a seed.
Output: Public key $\mathbf{h}$ and secret key $(\mathbf{f}, \mathbf{g})$.
1: Instantiate Sampler with $\mathcal{X}_\sigma^N$ and seed
2: $\mathbf{f} \leftarrow$ Sampler, $\mathbf{g} \leftarrow$ Sampler
3: $\mathbf{h} = \mathbf{g}/(p\mathbf{f} + 1) \mod q$
```

We show in Algorithm 7 the encryption algorithm of ss-ntru-pke and in Algorithm 8 the decryption algorithm, both from the original proposal [Zha+17]. In these descriptions, Hash() represents a hash function, and $\mathcal{B}$ represents a set including all binary polynomials with degree at most $N - 1$. The Pad() operation is a function to ensure the message has sufficient entropy, and the Extract() operation is the inverse of Pad().

In each encryption of a message $\mathbf{m}$, two polynomials $\mathbf{r}, \mathbf{e} \in \mathcal{R}$ are generated, where the coordinates are again chosen from a discrete Gaussian distribution $\mathcal{X}_\sigma$ with standard deviation σ. This randomness source uses a seed generated as $\text{Hash}(\mathbf{m}, \mathbf{h})$. This means that each choice of a message $\mathbf{m}$ will generate also the polynomials $\mathbf{r}, \mathbf{e} \in \mathcal{R}$. Let us denote this by

$$(\mathbf{r}, \mathbf{e}) = \mathbf{G}(\mathbf{m}, \mathbf{h}).$$

Algorithm 7 ss-ntru-pke.ENCRYPT

```
Input: Public key $\mathbf{h}$, message $msg$ of length $mlen$, PARAM and a $seed$.
Output: Ciphertext $\mathbf{c}$.
$\mathbf{m}$ = Pad($msg$, $seed$)
$rseed$ = HASH($\mathbf{m}|\mathbf{h}$)
Instantiate Sampler with $\mathcal{X}_\sigma^N$ and $rseed$
$\mathbf{r} \leftarrow$ Sampler, $\mathbf{e} \leftarrow$ Sampler
$\mathbf{t} = p \cdot \mathbf{r} * \mathbf{h}$
$tseed$ = HASH($\mathbf{t}$)
Instantiate Sampler with $\mathcal{B}$ and $tseed$
$\mathbf{m}_{mask} \leftarrow$ Sampler
$\mathbf{m}' = \mathbf{m} - \mathbf{m}_{mask} \pmod p$
$\mathbf{c} = \mathbf{t} + p \cdot \mathbf{e} + \mathbf{m}'$
```

In decryption, with ciphertext $\mathbf{c}$, one computes the message by computing

$$\mathbf{f} * \mathbf{c} = p \cdot \mathbf{r} * \mathbf{g} + p \cdot \mathbf{e} * \mathbf{f} + \mathbf{m}' * \mathbf{f}.$$

A decryption error occurs if $||p \cdot \mathbf{r} * \mathbf{g} + p \cdot \mathbf{e} * \mathbf{f} + \mathbf{m}' * \mathbf{f}||_\infty > q/2$. This basically translates to $||\mathbf{r} * \mathbf{g} + \mathbf{e} * \mathbf{f}||_\infty > q/4$ as $p = 2$ and the last term is much smaller than the first two.

The proposed parameters for ss-ntru-pke for the security level of NIST-V are shown in Table 2. The decoding error probability is estimated to be less than 2^{-80} in [Zha+17].

Table 2: Proposed ss-ntru-pke parameters.

N	q	p	$\mathcal{R}$	σ	ϵ	Security
1024	$2^{30} + 2^{13} + 1$	2	$\frac{\mathbb{Z}_q[x]}{x^N+1}$	724	$< 2^{-80}$	V

7.2 The attack

We now follow the approach of the previous section and describe an attack. The detailed attack is shown in Algorithm 9, where a more efficient CCA2 version is adopted. We define an equivalence relation for two polynomials $u(x), v(x) \in \mathcal{R}$ if $u(x) = x^i \cdot v(x) \pmod{x^N + 1}$, or if $u(x) = -x^i \cdot v(x) \pmod{x^N + 1}$, for $i \in \mathbb{Z}$.

Algorithm 8 ss-ntru-pke.DECRYPT

```
Input: Secret key $\mathbf{f}$, public key $\mathbf{h}$, ciphertext $\mathbf{c}$, and PARAM.
Output: result.
$\mathbf{m}' = \mathbf{f} * \mathbf{c} \pmod{p}$;
$\mathbf{t} = \mathbf{c} - \mathbf{m}'$;
tseed = HASH($\mathbf{t}$);
Instantiate Sampler with $\mathcal{B}$ and tseed;
$\mathbf{m}_{mask} \leftarrow$ Sampler;
$\mathbf{m} = \mathbf{m}' + \mathbf{m}_{mask} \pmod{p}$;
rseed = HASH($\mathbf{m}|\mathbf{h}$);
Instantiate Sampler with $\mathcal{X}_\sigma^N$ and rseed;
$\mathbf{r} \leftarrow$ Sampler;
$\mathbf{e} = p^{-1}\,(\mathbf{t} - \mathbf{r} * \mathbf{h})$;
if $||\mathbf{e}||_\infty$ is big then
    result = $\perp$;
else
    result = Extract($\mathbf{m}$);
end if
```

Algorithm 9 The CCA2 attack against ss-ntru-pke

```
Input: A number (say $2^{64}$) of public keys.
Output: The secret polynomials $(\mathbf{f}, \mathbf{g})$ of one public key.
(Attack step 1): Collect messages/ciphertexts with special form for all public keys
(Attack step 2): Submit them for decryption and determine a weak public key $\mathbf{h}$
(Attack step 1'): Prepare messages/ciphertexts with special form for this weak key $\mathbf{h}$
(Attack step 2'): Submit them for decryption and collect the decryption results
Use statistical analysis to guess $(\hat{\mathbf{f}}, \hat{\mathbf{g}})$ close to the corresponding secret key $(\mathbf{f}, \mathbf{g})$
Use lattice reduction algorithms to recover the secret key $(\mathbf{f}, \mathbf{g})$
```

Attack step 1 – pre-computation.

We pick random messages $\mathbf{m}$ and generate corresponding $(\mathbf{r}, \mathbf{e}) = \mathbf{G}(\mathbf{m}, \mathbf{h})$ for a given public key $\mathbf{h}$. We keep only vectors $\mathbf{e}$ equivalent to a polynomial that has the first l (e.g., $l = 2$) positions with the same sign and each with size larger than $c \cdot \sigma$, where c is a constant determining the computational effort of finding such error vectors. These vectors form our chosen set $\mathcal{F}$.

We set $l = 2$ to illustrate the idea in a concrete attack. For one position, the probability that the entry is larger than $c\sigma$ is $1 - \text{erf}(c/\sqrt{2})$. As we can start from any position, the probability to have two consecutive positions with the same sign and entries larger than $c\sigma$ is $p_{\text{e}} = N * (1 - \text{erf}(c/\sqrt{2}))^2/2$. If we set p_{e} to be 2^{-120}, then c can be as large as 9.193.

Attack step 2 – submit ciphertexts for decryption.

We then send the ciphertexts corresponding to the noise vectors in $\mathcal{F}$ to the decryption algorithm. If the targeted secret key $\mathbf{f}$ is also equivalent to a polynomial that has the first l (e.g., $l = 2$) positions with the same sign and each with size larger than $c_s \cdot \sigma$, where c_s is another constant, then the decoding errors can be detectable. We expect to collect several errors and store their corresponding error vectors $(\mathbf{r}, \mathbf{e})$. The probability to have two consecutive positions with the same sign and entries larger than $c_s\sigma$ is $p_{\text{s}} = N * (1 - \text{erf}(c_{\text{s}}/\sqrt{2}))^2/2$. If we set p_{s} to be 2^{-64}, then c_{s} can be as large as 6.802.

If we run 2^{120} precomputation steps for each stored vector with the desired properties, then the overall complexity is 2^{248} since $p_{\text{s}} = 2^{-64}$. Let C_1 denote $2 \cdot c_{\text{s}} c\sigma^2$. We can then have a coefficient in $\mathbf{r} * \mathbf{g} + \mathbf{e} * \mathbf{f}$ whose absolute contribution from these two big entries is at least $C_1 = 2^{25.97}$. We consider the probabilistic behavior of the remaining $(2N - 2)$ positions. As the coefficients of $\mathbf{r}, \mathbf{g}, \mathbf{e}, \mathbf{f}$ are all sampled from a Gaussian distribution with mean 0 and stand deviation $\sigma = 724$, the expected norm of the rest vector in $\mathbf{f}, \mathbf{g}$ with $2N - 2$ entries is about $\sqrt{2N-2} \cdot \sigma$. Given a public key, $\mathbf{f}, \mathbf{g}$ is fixed. Thus, this coefficient of $\mathbf{r} * \mathbf{g} + \mathbf{e} * \mathbf{f}$ can be approximated as $C_1 + \Phi_0$, where Φ_0 is Gaussian distribution with mean 0 and standard deviation $\sqrt{2N-2} \cdot \sigma^2$. As the error appears when this coefficient is larger than $q/4$, the error probability[3] can be approximated as

$$P_e = \left(1 - \text{erf}\left(\frac{q/4 - C_1}{\sqrt{2(2N-2)\sigma^2}}\right)\right) \cdot \frac{1}{2}.$$

We obtain a decoding error probability of $2^{-57.3}$ for this example.

Thus we can obtain about $2^{6.7}$ errors from the 2^{64} decryption trails.

[3] The error can occur in both directions. We omit the term $\left(1 - \text{erf}\left(\frac{q/4+C_1}{\sqrt{2(2N-2)\sigma^2}}\right)\right) \cdot \frac{1}{2}$ as it is negligible compared with $\left(1 - \text{erf}\left(\frac{q/4-C_1}{\sqrt{2(2N-2)\sigma^2}}\right)\right) \cdot \frac{1}{2}$ for C_1 a very big positive integer.

An adaptive CCA attack. If we keep the previous setting, i.e., a CCA1 attack, the cost is larger than 2^{248}. However, we can adopt a much more powerful attack model, namely an adaptive CCA (CCA2) attack, consisting of two phases. In the first phase, the attacker spends half of his computational power to determine a weak key; in the later phase, he would put all his remaining resources into attacking this weak key.

To be more specific, we first prepare 2^{63} messages/ciphertexts for each of the 2^{64} public keys. Then we expect two errors corresponding to one key, which can be claimed as a weak key.

We can also reduce the precomputation work for each key to 2^{89}, if there are 2^{64} public keys. We have $c = 7.956$ and the error probability is $2^{-62.0}$, so we expect to have two errors in the testing stage. We then spend 2^{216} work on another precomputation to have 2^{63} messages with c to be 10.351, done only for this weak key. The error probability in the second phase is estimated as $2^{-53.0}$, so we can have 2^{10} errors. The overall complexity is 2^{217}.

Attack step 3 – statistical analysis.

In this step we will try to recover the secret **f**. Let us first assume that **f** has its two big entries in the first two positions of the vector. Then the position in $\mathbf{e} * \mathbf{f}$ where the error occurs, denoted i_0, is the position where the two significant coefficients in **e** and those in **f** coincide. We now transform each **e** in such a way that its two big entries are also to be found in the first two positions. This is done by replacing **e** with the corresponding equivalent vector where the two big entries are in the first two positions. Assuming M decryption errors, this now gives us the following knowledge from the received decryption errors:

$$\sum_{i=2}^{N-1} e_i^{(j)} f_i + N_i^{(j)} > q/4 - 2 \cdot c_s c \sigma^2,$$

for $j = 1 \dots M$ and where $N^{(j)}$ denotes the remaining contribution to the noise. Finally we note that assuming that **f** has its two big entries in the first two positions is not a restriction, as such an **f** vector will just be an equivalent vector of the true **f**. So we need only to recover **f** and then check all equivalent vectors.

We next show how to derive more knowledge of **f**, **g** with statistical tools.

A heuristic approach. As we have assumed that the two big entries in $(\mathbf{f}, \mathbf{g})$ (or $(\mathbf{e}, \mathbf{r})$) are the first two entries, we use **K** (or $\mathbf{V}_i$ for $1 \leq i \leq M$) to denote a vector consisting of the remaining $2N - 2$ entries. Thus, the size of **K** (or $\mathbf{V_i}$) can be estimated as $\sqrt{(2N-2)}\sigma$.

We adopt the heuristic assumptions from [GN07] that all the errors are very close to the folding bound $q/4$, meaning that all the messages leading to an error belong to a hyperplane

$$\mathbf{V}_i \cdot \mathbf{K} = \frac{q}{4} - C_1,$$

where C_1 is the contribution from the two significant entries.

Thus, the mean vector $\hat{\mathbf{V}}$ of $\mathbf{V_i}$ should be close to a scaled vector of $\mathbf{K}$, i.e.,

$$\hat{\mathbf{V}} = \frac{\sum_{i=1}^{M} \mathbf{V}_i}{M} \approx \frac{q/4 - C_1}{||\mathbf{K}||^2}\mathbf{K}.$$

We can have an estimation $\hat{\mathbf{K}} = \frac{(2N-2)\sigma^2}{q/4-C_1}\hat{\mathbf{V}}$. If we round the entries of $\mathbf{K}$ to the nearest integer in $\mathbb{Z}_q$, we obtain an estimation $(\hat{\mathbf{f}}, \hat{\mathbf{g}})$ of the secret vector $(\mathbf{f}, \mathbf{g})$.

The remaining question is how good this estimation can be? We heuristically answer this question using the central limit theorem.

Each observation $\mathbf{V_i}$ with approximated norm $\sqrt{2N-2}\sigma$ can be viewed as the summation of the signal point

$$\frac{q/4 - C_1}{||\mathbf{K}||^2}\mathbf{K},$$

and a noise vector with squared norm

$$(2N-2)\sigma^2 - \frac{(q/4 - C_1)^2}{(2N-2)\sigma^2}.$$

By the central limit theorem, if we have M observations, then the squared norm (variance) of the noise can be reduced by a factor of M. Hence, the error norm should be

$$\sqrt{\frac{1}{M} \cdot \left((2N-2)\sigma^2 - \frac{(q/4 - C_1)^2}{(2N-2)\sigma^2}\right)}.$$

As we consider $\hat{\mathbf{K}}$ instead of $\hat{\mathbf{V}}$, the true error norm should be resized as

$$\frac{(2N-2)\sigma^2}{q/4 - C_1} \cdot \sqrt{\frac{1}{M} \cdot \left((2N-2)\sigma^2 - \frac{(q/4 - C_1)^2}{(2N-2)\sigma^2}\right)}. \tag{29}$$

Using this formula, we can have a candidate with error norm $0.169\sqrt{2N-2}\sigma$, assuming that 1024 errors have been collected.

Attack step 4 – lattice reduction.

If $(\Delta\mathbf{f}, \Delta\mathbf{g}) = (\mathbf{f}, \mathbf{g}) - (\hat{\mathbf{f}}, \hat{\mathbf{g}})$ is small, we can recover it using lattice reduction algorithms efficiently. Thus, we obtain the correct value of $(\mathbf{f}, \mathbf{g})$.

If we have the error size to be only $0.169\sqrt{2N-2}\sigma$, as assumed in the previous step, using the LWE estimator from Albrecht et al. [Alb+18], it takes about 2^{181} time and 2^{128} memory if one uses sieving to implement the SVP oracle in BKZ. Though the authors of [Zha+17] discussed about memory constraint for applying sieving in lattice-based cryptanalysis, we believe it is reasonable to assume for 2^{128} memory when considering a scheme targeting the classic 256-bit security level. Another possibility is to implement the SVP oracle using tuple sieving, further reducing the memory complexity to 2^{117}. The time complexity then increases to 2^{223}, but still far from achieving the claimed 256-security level.

7.3 Experimental results

Table 3: The simulated error rates v.s. the estimated error rates.

q	error rate	
	-estimated-	-simulated-
$q = 2^{29}$	$2^{-9.05}$	$2^{-9.19}$
$q = 2^{29} + 2^{26}$	$2^{-12.64}$	$2^{-12.96}$
$q = 2^{29} + 2^{27}$	$2^{-16.91}$	$2^{-17.09}$
$q = 2^{29} + 2^{27} + 2^{26} + 2^{25}$	$2^{-24.62}$	$2^{-24.57}$

We have implemented some of the parts of the attack to check performance against theory. We have chosen exactly the same parameters in ss-ntru-pke as well as in the attack, except for the q value, which in the experiment was set to the values shown in Table 3. The reason being that is we wanted to lower the decryption error rate so that simulation was possible.

We put two consecutive entries in **f** each of size $6.2 \cdot \sigma$ and we generated error vectors with two large positive entries each of size $9.2 \cdot \sigma$. For such choice, we first verified the decryption error probabilities, as seen in Table 3. These match the theoretical results well.

Table 4: The simulated error norm v.s. the estimated error norm. ($M = 1024$)

q	error norm $/(\sqrt{2N-2}\sigma)$	
	-estimated-	-simulated-
$q = 2^{29}$	0.487	0.472
$q = 2^{29} + 2^{26}$	0.391	0.360
$q = 2^{29} + 2^{27}$	0.326	0.302
$q = 2^{29} + 2^{27} + 2^{26} + 2^{25}$	0.261	0.250

For each choice of q we then collected up to $M = 2^{10} + 2^9 = 1536$ error vectors and processed them in a statistical analysis step, to get a good approximation of $(\mathbf{f}, \mathbf{g})$. As the heuristic approach described, we first created an approximation of $(\mathbf{f}, \mathbf{g})$, say denoted by $(\hat{\mathbf{f}}, \hat{\mathbf{g}})$, by simply computing $\hat{f}_i = E \cdot \frac{\sum_{j=0}^{M-1} e_i^{(j)}}{M}$ as the

Table 5: The simulated error norm v.s. the estimated error norm. ($q = 2^{29} + 2^{27} + 2^{26} + 2^{25}$)

M	error norm $/(\sqrt{2N-2}\sigma)$	
	-estimated-	-simulated-
$M = 256$	0.522	0.490
$M = 512$	0.369	0.348
$M = 1024$	0.261	0.250
$M = 1536$	0.213	0.212

value in the ith position. Here E is a constant that makes the norm of the vector to be as the expected norm of **f**. Clearly, this is a very simple way of exploring the dependence between f_i and e_i, but still it seems to be sufficient.

We have plotted the simulated error norms for various q and M in Fig. 9. Furthermore, we show in Table 4 and Table 5 the comparison between the simulated error norms and the estimated error norms according to Eq. (29).

In the prior table, M is fixed to 1024 and q varies, while in the latter table, q is fixed to $2^{29} + 2^{27} + 2^{26} + 2^{25}$ and M varies. We see that in all the cases, the simulated data match the estimated data well, though the simulation seems always better than the estimation, i.e., with smaller error norms. Another observation from Table 5 is that the estimation using the central limit theorem becomes more accurate when M becomes larger, which is also very reasonable.

7.4 Summarizing the attack

The best attack is a CCA2 type attack where we in precomputation use $2^{89+63} = 2^{152}$ operations to derive 2^{63} special ciphertexts that are submitted for decryption. With probability 2^{-64} the secret **f** has the desired property of two consecutive big entries. If so, we will most likely see several decoding errors and such a weak key has been detected. When the weak key has been detected, we perform yet another precomputation that uses 2^{216} operations to derive 2^{63} additional special ciphertexts again submitted for decryption. We receive in expectation 1024 decryption errors and the knowledge from the error vectors will allow us to reconstruct **f** without too much trouble using lattice reduction algorithms, as experimental results strongly indicated. The overall complexity is thus approximately 2^{217} if the SVP oracle in BKZ is implemented via lattice sieving. Actually, the cost of the lattice reduction algorithms in the final stage is not the bottleneck, since we can employ other powerful statistical tools in Step 3 (e.g., the Maximum Likelihood Test approach) to make this cost negligible.

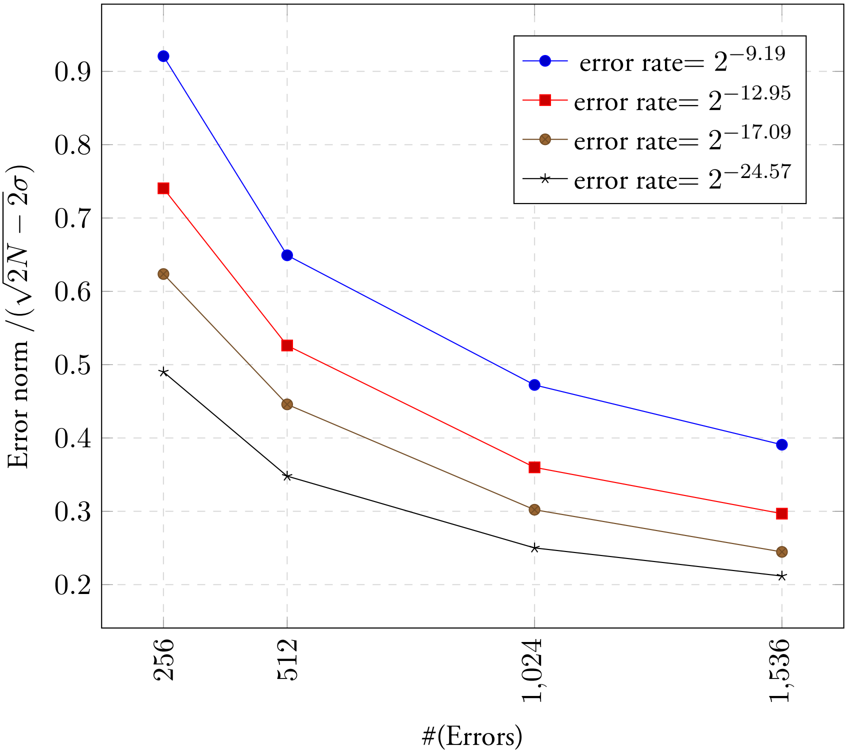


Figure 9: Error norm as a function of the number of collected error vectors.

8 Conclusion

In this paper we introduced a method to increase the decryption failure rate of a scheme, based on the search for 'weak' ciphertexts. This method benefits an adversary in at least three scenarios: if he has access to a quantum computer, if he can only perform a limited number of decryption queries or if he wants to stage a multi-target attack on schemes that do not have the appropriate protection. We explicitly calculated the effect of failure boosting in these scenarios for various (Ring/Module) LWE/LWR schemes. We also proposed a method to estimate the secret key given ciphertexts that lead to decryption failures. The remaining security after a certain number of decryption failures was calculated, given the exact location of the error. We suggested three methods to obtain the exact location of errors in failing ciphertexts. Finally, we estimated the security of several schemes under an attack that optimally uses these decryption failures and show that for some schemes the security is drastically reduced if an attacker can perform sufficient decryption queries. However, for most NIST post-quantum standardization candidates, the expected number of required decryption queries is too high for a practical attack. We also identify the changes to this attack under a multi-target scenario or when an attacker has only access to a limited number of decryption queries.

We further proposed a generic weak-key attack model against lattice-based schemes, which is slightly different from the previous attack, based on the observation that the error probability can be much higher for certain 'weak' keys. We applied this model to attacking ss-ntru-pke, a version in the NTRUEncrypt submission to the NIST Post-Quantum Standardization Process. Specifically, we have presented an adaptive CCA attack on the claimed 256-bit classic security level (NIST-V) of ss-ntru-pke. This attacking idea can be treated as extension of reaction attacks [GJS16; Fab+17] that already jeopardize the CCA security of MDPC and LDPC based crypto-systems.

9 Acknowledgements

The authors would like to thank Tancrède Lepoint and the anonymous reviewers for their helpful comments. They would also like to thank Andreas Hülsing for interesting discussions. This work was supported in part by the Research Council KU Leuven: C16/15/058, by the European Commission through the Horizon 2020 research and innovation programme Cathedral ERC Advanced Grant 695305, by the Research Council KU Leuven grants C14/18/067 and STG/17/019, by the Norwegian Research Council (Grant No. 247742/070), by the Swedish Research Council (Grant No. 2015-04528), by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, and by the Swedish Foundation for Strategic Research (SSF) project RIT17-0005.

References

[Alb+18] M. R. Albrecht et al. *Estimate all the LWE, NTRU schemes!* Cryptology ePrint Archive, Report 2018/331. https://eprint.iacr.org/2018/331. 2018.

[Alk+16] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. "Post-quantum key exchange – a new hope". In: *USENIX Security 2016*. 2016.

[APS15] M. Albrecht, R. Player, and S. Scott. *On the concrete hardness of Learning with Errors*. Journal of Mathematical Cryptology. Oct. 2015.

[Baa+17] H. Baan et al. *Round2: KEM and PKE based on GLWR.* Cryptology ePrint Archive, Report 2017/1183. https://eprint.iacr.org/2017/1183. 2017.

[Ber+18] D. J. Bernstein, L. G. Bruinderink, T. Lange, and L. Panny. "HILA5 Pindakaas: On the CCA Security of Lattice-Based Encryption with Error Correction". In: *AFRICACRYPT 18: 10th International Conference on Cryptology in Africa*. Ed. by A. Joux, A. Nitaj, and T. Rachidi. Vol. 10831. Lecture Notes in Computer Science. Marrakesh, Morocco: Springer, Heidelberg, Germany, May 2018, pp. 203–216.

[Bol+14] A. Boldyreva, J. P. Degabriele, K. G. Paterson, and M. Stam. "On Symmetric Encryption with Distinguishable Decryption Failures". In: *Fast Software Encryption – FSE 2013*. Ed. by S. Moriai. Vol. 8424. Lecture Notes in Computer Science. Singapore: Springer, Heidelberg, Germany, Mar. 2014, pp. 367–390.

[Bos+17] J. Bos et al. *CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM*. Cryptology ePrint Archive, Report 2017/634. `http://eprint.iacr.org/2017/634`. 2017.

[BPR12] A. Banerjee, C. Peikert, and A. Rosen. "Pseudorandom Functions and Lattices". In: *Advances in Cryptology – EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 719–737.

[Bra+13] Z. Brakerski, A. Langlois, C. Peikert, O. Regev, and D. Stehlé. "Classical hardness of learning with errors". In: *45th Annual ACM Symposium on Theory of Computing*. Ed. by D. Boneh, T. Roughgarden, and J. Feigenbaum. Palo Alto, CA, USA: ACM Press, June 2013, pp. 575–584.

[Che+16] J. H. Cheon, D. Kim, J. Lee, and Y. Song. *Lizard: Cut off the Tail! Practical Post-Quantum Public-Key Encryption from LWE and LWR*. Cryptology ePrint Archive, Report 2016/1126. `http://eprint.iacr.org/2016/1126`. 2016.

[DAn+18] J.-P. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren. "Saber: Module-LWR Based Key Exchange, CPA-Secure Encryption and CCA-Secure KEM". In: *AFRICACRYPT 2018*. 2018, pp. 282–305.

[Din+16] J. Ding, S. Alsayigh, S. RV, S. Fluhrer, and X. Lin. *Leakage of Signal function with reused keys in RLWE key exchange*. Cryptology ePrint Archive, Report 2016/1176. `http://eprint.iacr.org/2016/1176`. 2016.

[DVV18] J.-P. D'Anvers, F. Vercauteren, and I. Verbauwhede. *On the impact of decryption failures on the security of LWE/LWR based schemes*. Cryptology ePrint Archive, Report 2018/1089. `https://eprint.iacr.org/2018/1089`. 2018.

[Fab+17] T. Fabsic, V. Hromada, P. Stankovski, P. Zajac, Q. Guo, and T. Johansson. *A Reaction Attack on the QC-LDPC McEliece Cryptosystem*. Cryptology ePrint Archive, Report 2017/494. `http://eprint.iacr.org/2017/494`. 2017.

[Flu16] S. Fluhrer. *Cryptanalysis of ring-LWE based key exchange with key share reuse*. Cryptology ePrint Archive, Report 2016/085. `https://eprint.iacr.org/2016/085`. 2016.

[FO99] E. Fujisaki and T. Okamoto. "Secure Integration of Asymmetric and Symmetric Encryption Schemes". In: *Advances in Cryptology – CRYPTO'99*. Ed. by M. J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1999, pp. 537–554.

[GJN19] Q. Guo, T. Johansson, and A. Nilsson. *A Generic Attack on Lattice-based Schemes using Decryption Errors with Application to ss-ntru-pke*. Cryptology ePrint Archive, Report 2019/043. `https://eprint.iacr.org/2019/043`. 2019.

[GJS16] Q. Guo, T. Johansson, and P. Stankovski. "A Key Recovery Attack on MDPC with CCA Security Using Decoding Errors". In: *Advances in Cryptology – ASIACRYPT 2016, Part I*. Ed. by J. H. Cheon and T. Takagi. Vol. 10031. Lecture Notes in Computer Science. Hanoi, Vietnam: Springer, Heidelberg, Germany, Dec. 2016, pp. 789–815.

[GN07] N. Gama and P. Q. Nguyen. "New Chosen-Ciphertext Attacks on NTRU". In: *PKC 2007: 10th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by T. Okamoto and X. Wang. Vol. 4450. Lecture Notes in Computer Science. Beijing, China: Springer, Heidelberg, Germany, Apr. 2007, pp. 89–106.

[Gro96] L. K. Grover. "A Fast Quantum Mechanical Algorithm for Database Search". In: *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*. STOC '96. New York, NY, USA: ACM, 1996, pp. 212–219.

[HGS99] C. Hall, I. Goldberg, and B. Schneier. "Reaction Attacks against several Public-Key Cryptosystems". In: *ICICS 99: 2nd International Conference on Information and Communication Security*. Ed. by V. Varadharajan and Y. Mu. Vol. 1726. Lecture Notes in Computer Science. Sydney, Australia: Springer, Heidelberg, Germany, Nov. 1999, pp. 2–12.

[HHK17] D. Hofheinz, K. Hövelmanns, and E. Kiltz. "A Modular Analysis of the Fujisaki-Okamoto Transformation". In: *TCC 2017: 15th Theory of Cryptography Conference, Part I*. Ed. by Y. Kalai and L. Reyzin. Vol. 10677. Lecture Notes in Computer Science. Baltimore, MD, USA: Springer, Heidelberg, Germany, Nov. 2017, pp. 341–371.

[How+03a] N. Howgrave-Graham, P. Q. Nguyen, et al. "The Impact of Decryption Failures on the Security of NTRU Encryption". In: *Advances in Cryptology – CRYPTO 2003*. Ed. by D. Boneh. Vol. 2729. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2003, pp. 226–246.

[How+03b] N. Howgrave-Graham, J. H. Silverman, A. Singer, and W. Whyte. *NAEP: Provable Security in the Presence of Decryption Failures*. Cryptology ePrint Archive, Report 2003/172. `http://eprint.iacr.org/2003/172`. 2003.

[HS00] J. Hoffstein and J. H. Silverman. "Protecting NTRU Against Chosen Ciphertext and Reaction Attacks". In: 2000.

[Jia+17] H. Jiang, Z. Zhang, L. Chen, H. Wang, and Z. Ma. *Post-quantum IND-CCA-secure KEM without Additional Hash*. Cryptology ePrint Archive, Report 2017/1096. `https://eprint.iacr.org/2017/1096`. 2017.

[JJ00] É. Jaulmes and A. Joux. "A Chosen-Ciphertext Attack against NTRU". In: *Advances in Cryptology — CRYPTO 2000*. Ed. by M. Bellare. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 20–35.

[LPR10] V. Lyubashevsky, C. Peikert, and O. Regev. "On Ideal Lattices and Learning with Errors over Rings". In: *Advances in Cryptology – EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 – June 3, 2010. Proceedings*. Springer Berlin Heidelberg, 2010, pp. 1–23.

[LS15] A. Langlois and D. Stehlé. "Worst-case to average-case reductions for module lattices". In: *Designs, Codes and Cryptography* 75.3 (June 2015), pp. 565–599.

[Lu+17] X. Lu, Y. Liu, D. Jia, H. Xue, J. He, and Z. Zhang. *Lac*. Technical report, National Institute of Standards and Technology. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`. 2017.

[Nae+17] M. Naehrig et al. *FrodoKEM*. Technical report, National Institute of Standards and Technology. available at `https://frodokem.org/files/FrodoKEM-specification-20171130.pdf`. 2017.

[NIS16] NIST. *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process.* `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf`. 2016.

[NIS19] NIST. *NIST Post-Quantum Cryptography Forum*. `https://groups.google.com/a/list.nist.gov/g/pqc-forum` Accessed: 2019-01-11. 2019.

[NJS19] A. Nilsson, T. Johansson, and P. Stankovski. "Error Amplification in Code-based Cryptography". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019.1 (2019), pp. 238–258.

[Pei09] C. Peikert. "Public-key Cryptosystems from the Worst-case Shortest Vector Problem: Extended Abstract". In: *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*. STOC '09. New York, NY, USA: ACM, 2009, pp. 333–342.

[Reg05] O. Regev. "On lattices, learning with errors, random linear codes, and cryptography". In: *37th Annual ACM Symposium on Theory of Computing*. Ed. by H. N. Gabow and R. Fagin. Baltimore, MA, USA: ACM Press, May 2005, pp. 84–93.

[Saa17] M.-J. O. Saarinen. *HILA5*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`. National Institute of Standards and Technology, 2017.

[Sch+17a] J. M. Schanck, A. Hulsing, J. Rijneveld, and P. Schwabe. *NTRU-HRSS-KEM*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`. National Institute of Standards and Technology, 2017.

[Sch+17b] P. Schwabe et al. *Crystals-Kyber*. Technical report, National Institute of Standards and Technology. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`. 2017.

[Sch+17c] P. Schwabe et al. *NewHope*. Technical report, National Institute of Standards and Technology. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`. 2017.

[Seo+17] M. Seo, J. H. Park, D. H. Lee, S. Kim, and S.-J. Lee. *Emblem and R.Emblem*. Technical report, National Institute of Standards and Technology. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`. 2017.

[Sma+17] N. P. Smart et al. *Lima*. Technical report, National Institute of Standards and Technology. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`. 2017.

[SS11] D. Stehlé and R. Steinfeld. "Making NTRU as Secure as Worst-Case Problems over Ideal Lattices". In: *Advances in Cryptology – EUROCRYPT 2011*. Ed. by K. G. Paterson. Vol. 6632. Lecture Notes in Computer Science. Tallinn, Estonia: Springer, Heidelberg, Germany, May 2011, pp. 27–47.

[SXY17] T. Saito, K. Xagawa, and T. Yamakawa. *Tightly-Secure Key-Encapsulation Mechanism in the Quantum Random Oracle Model*. Cryptology ePrint Archive, Report 2017/1005. `https://eprint.iacr.org/2017/1005`. 2017.

[Sze17] A. Szepieniec. *Ramstake*. Technical report, National Institute of Standards and Technology. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`. 2017.

[TU16] E. E. Targhi and D. Unruh. "Post-Quantum Security of the Fujisaki-Okamoto and OAEP Transforms". In: *Theory of Cryptography: 14th International Conference, TCC 2016-B, Beijing, China, October 31-November 3, 2016, Proceedings, Part II*. Springer Berlin Heidelberg, 2016, pp. 192–216.

[Zha+17] Z. Zhang, C. Chen, J. Hoffstein, and W. Whyte. *NTRUEncrypt*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`. National Institute of Standards and Technology, 2017.

A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM

Abstract

In the implementation of post-quantum primitives, it is well known that all computations that handle secret information need to be implemented to run in constant time. Using the Fujisaki-Okamoto transformation or any of its different variants, a CPA-secure primitive can be converted into an IND-CCA secure KEM. In this paper we show that although the transformation does not handle secret information apart from calls to the CPA-secure primitive, it has to be implemented in constant time. Namely, if the ciphertext comparison step in the transformation is leaking side-channel information, we can launch a key-recovery attack.

Several proposed schemes in round 2 of the NIST post-quantum standardization project are susceptible to the proposed attack and we develop and show the details of the attack on one of them, being FrodoKEM. It is implemented on the reference implementation of FrodoKEM, which is claimed to be secure against all timing attacks. Experiments show that the attack code is able to extract the secret key for all security levels using about 2^{30} decapsulation calls.

Q. Guo, T. Johansson, and A. Nilsson. "A Key-Recovery Timing Attack on Post-quantum Primitives Using the Fujisaki-Okamoto Transformation and Its Application on FrodoKEM". in: *Advances in Cryptology – CRYPTO 2020*. Ed. by D. Micciancio and T. Ristenpart. Cham: Springer International Publishing, 2020, pp. 359–386

1 Introduction

Post-Quantum Cryptography is the area of cryptographic research in the presence of an, assumed to be practical, quantum computer. It is well known that most of today's public-key solutions are insecure under this assumption since they are based on the difficulty of factoring or the discrete log problem. These two problems can be solved in polynomial time if a large enough quantum computer exists [Sho94]. Instead, post-quantum cryptography is based on other hard problems, not known to be broken by a quantum computer. The two most popular areas are lattice-based schemes and code-based schemes.

Learning with errors (LWE) is a hard problem that is closely connected to difficult problems in lattices, such as the shortest vector problem. Learning with errors, or some version of the problem, is used in many of the recently proposed schemes to build public-key encryption schemes (PKE) and key encapsulation mechanisms (KEM).

Code-based schemes are similar to LWE schemes, but rely instead on difficult coding theory problems, like finding a minimum (Hamming) weight codeword in a binary code. Code-based schemes date back to 1978 and the McEliece PKE scheme [McE78].

The importance of post-quantum cryptography is highlighted by the fact that NIST is currently running a standardization project on post-quantum cryptography [NIS18] which is in the second round. Most KEM- and PKE-schemes remaining in the NIST post-quantum cryptography standardization project (NIST PQ project) are either lattice-based or code-based schemes.

A very common approach for the above mentioned types of schemes is to construct a public-key encryption scheme that is secure in the chosen plaintext model (CPA) and then use a generic transformation to transform the scheme into a IND-CCA secure KEM. An IND-CCA secure primitive is secure in in the model of indistinguishability under chosen ciphertext attacks. For definitions of these models, we refer to any textbook on the subject [Sma16]. The most common generic transformation is the Fujisaki-Okamoto (FO) transformation [FO99] or any of its many different variants [HHK17]. It gives IND-CCA security in the random oracle model, and there is also a post-quantum secure version [HHK17]. This is also the way lattice-based and code-based KEM schemes in the NIST PQ project are constructed. They all use some version of the FO transformation.

In the implementation of post-quantum primitives, it is well known that all computations that handle secret information need to be implemented to run in constant time. Leakage of timing information can give information about secret values. This is a hard problem in practice, as we might not trust just any programmer to pay enough attention to such issues. There is now much focus on constant time implementations for the remaining candidates in the NIST PQ project and much research is devoted to examine cryptanalysis on so called side-channels [BB03; Koc96]. This includes work showing attacks on schemes with implementations that leak timing information when processing secret data, such as the step of decoding an error correcting code inside the decryption scheme.

In this paper we show that even though the FO transformation itself does not handle secret information apart from calls to the CPA-secure PKE (running in constant time), it still has to be implemented in constant time. Namely, if the ciphertext comparison step in the FO transformation is leaking side-channel information, we can launch a key-recovery attack. The attack is based on generating decryption failures in the CPA-secure primitive by modifying the ciphertext. Through timing information we can learn whether a modified ciphertext is decrypted to the same message as the original ciphertext, or not.

This kind of attack has not been observed before, as several of the NIST candidates provide implementations that are directly susceptible to the proposed attack. We mention that at least the round 2 candidates FrodoKEM, LAC, BIKE, HQC, ROLLO and RQC have all submitted reference implementations that potentially leaked timing information in the ciphertext comparison step, and thus they might be susceptible to this attack.

We decided to develop and show the details of the attack on one of them, being FrodoKEM. FrodoKEM is a lattice-based KEM where the security is based on the LWE problem. It is a very conservative design with strong security proofs and its design team contains many very distinguished cryptographers. In the document [Nae+19] submitted to NIST, it is claimed that "All our implementations avoid the use of secret address accesses and secret branches and, hence, are protected against timing and cache attacks." An implementation of FrodoKEM that can be attacked also appears in Open Quantum Safe [OQS20].

The attack on FrodoKEM is detailed in the paper and then implemented on the reference implementation of FrodoKEM. Using experiments, we show that the attack code, by measuring the execution time of full decapsulations, is able to collect enough data for a complete secret key recovery using about 2^{30} decapsulation queries. We target the FrodoKEM parameters for the highest security level.

Previous work: Some previous work on using decryption failures in cryptanalysis appeared in [How+03a; How+03b]. More recent attacks using decryption failures on lattice-based schemes are to be found in [Bae+19; Bau+19; Ber+17; Flu16]. None of these attacks apply to CCA-secure schemes unless there is some misuse of the scheme.

Attacks on CCA secure schemes based on failures were modelled in [DAn+19a] and a complex attack on an NTRU version was presented. An attack on LAC [Lu+19] using decryption errors was given in [GJY19].

Side-channel attacks were first proposed by Kocher in [Koc96]. In such an attack, information obtained through physical measurements is used in the attack, being timing information, power measurements, electromagnetic radiation or other means. Brumley and Boneh attacked OpenSSL in [BB03], showing that remote timing attacks are practical. Side-channel attacks on post-quantum primitives have been proposed on signature schemes like BLISS [Bru+16]. Side-channel attacks on encryption/KEM primitives have been less successful, but include [Fac+18] measuring the robustness of the candidates in the NIST PQ project against cache-timing attacks. An attack on LWE schemes that use error correcting codes for which decoding is not implemented in constant time was given in [DAn+19b] and the same for code-based schemes are given in [Str10] and [Str13].

Paper organization: The remaining parts of the paper are organized as follows. Section Section 2 gives basic notation and definitions used later in the paper. In Section Section 3 we give a high-level description of the attack and describe the general underlying ideas used to achieve success in building the key parts of the attack. In Section Section 4 we describe the FrodoKEM scheme and briefly highlight the weakness in its reference implementation. In Section Section 5 we give the full details on how to apply the attack on FrodoKEM and recover the secret key. Results on implementing the attack on the FrodoKEM reference implementation are given. Finally, we discuss in Section Section 6 a few other round 2 NIST schemes where the reference implementations can be attacked, including LWE schemes using error correction and pure code-based schemes. Further details on how the attack could be adapted to LAC is found in the appendix.

2 Preliminary

We start by defining some useful notations used throughout the rest of the paper. In post-quantum cryptography with emphasis on lattice-based or code-based schemes, it makes sense to consider the message $\mathbf{m} \in \mathcal{M}$ and ciphertext $\mathbf{c} \in \mathcal{C}$ as being vectors with entries in some alphabet $\mathbb{Z}_q$. A PKE is then a triple of algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, where KeyGen generates the secret key $\mathsf{sk} \in \mathcal{SK}$ and the public key $\mathsf{pk} \in \mathcal{PK}$. The encryption algorithm Enc maps the message to a ciphertext using the public key and the decryption algorithm Dec maps the ciphertext back to a message using the secret key. Encryption may also use some randomness denoted $\mathbf{r} \in \mathcal{R}$, that can be viewed as part of the input to the algorithm. If $\mathbf{c} = \mathsf{Enc}(\mathsf{pk}, \mathbf{m}; \mathbf{r})$, then decrypting such a ciphertext, i.e., computing $\mathsf{Dec}(\mathsf{sk}, \mathsf{pk}, \mathbf{c})$, returns the ciphertext $\mathbf{m}$. Some schemes may have a small failure probability, meaning that the decryption algorithm fails to return a correctly encrypted message when decrypted.

A KEM is similarly defined as a triple of algorithms (KeyGen, Encaps, Decaps), where KeyGen generates the secret key $\mathsf{sk} \in \mathcal{SK}$ and the public key $\mathsf{pk} \in \mathcal{PK}$. The encapsulation algorithm Encaps generates a random session key, denoted as $\mathbf{s} \in \mathcal{S}$, and computes a ciphertext $\mathbf{c}$ using the public key. Applying the decapsulation algorithm Decaps on a ciphertext using the secret key returns the chosen session key $\mathbf{s}$, or possibly a random output in case the ciphertext does not fully match a possible output of the encapsulation algorithm.

Security can be defined in many different models, but most commonly the analysis is done in the CPA model, where the adversary essentially only have access to the public key pk and the public encryption/encapsulation calls. In a CCA model, the adversary is allowed to ask for decryptions/decapsulations of her choice. So for example, the notion of IND-CCA for a KEM is defined through the following game: Let the ciphertext $\mathbf{c}$ be the encapsulation of the secret key $\mathbf{s}_0 \in \mathcal{S}$. Consider another randomly selected key $\mathbf{s}_1 \in \mathcal{S}$. The adversary gets the ciphertext $\mathbf{c}$ as well as $\mathbf{s}_b$, where $b \in \{0, 1\}$ is randomly chosen. The adversarial task is to correctly predict b with as large probability as possible and access to the decapsulation oracle is allowed for all ciphertext inputs except $\mathbf{c}$. For more detailed definitions of these different security models, we refer to any textbook on the subject, for example [Sma16].

A very common approach is to construct a public-key primitive that is secure in the CPA model and then use a generic transformation to transform the scheme into a IND-CCA secure primitive. A common such generic transformation is the FO transformation [FO99], which has also many different variants [HHK17]. It gives IND-CCA security in the random oracle model and include a post-quantum secure version [HHK17]. This is also the way lattice-based and code-based KEM schemes in the NIST PQ project are constructed. They use some version of the FO transformation. We will introduce and investigate the FO transformation in relation to side-channel leakage in the next section.

3 A general description of the proposed attack

We describe the attack for post-quantum primitives in the form of KEMs, although the attack could work for other types of PKC primitives as well.

Let $\mathsf{PKE.CPA.Enc}(\mathsf{pk}, \mathbf{m}; \mathbf{r})$ denote a public key encryption algorithm which is secure in the CPA model. Here pk is the public key, $\mathbf{m}$ is the message to be encrypted, and $\mathbf{r}$ is the randomness used in the scheme. The algorithm returns a ciphertext $\mathbf{c}$. Furthermore, let $\mathsf{PKE.CPA.Dec}(\mathsf{sk}, \mathsf{pk}, \mathbf{c})$ denote the corresponding decryption algorithm. This algorithm returns a message, again denoted $\mathbf{m}$.

We will now assume that the $\mathsf{PKE.CPA.Dec}(\cdot)$ call is implemented in constant-time and is not leaking any side-channel information.

The CCA-secure KEM is assumed to be obtained through some variant of the FO transformation, resulting in algorithms for encapsulation and decapsulation similar to algorithms Algorithm 1 and Algorithm 2 shown here.

Here $\mathbf{k} \in \mathcal{K}$ and H_1 and H_2 are pseudo-random functions generating values indistinguishable from true randomness, with images $\mathcal{R} \times \mathcal{K}$ and $\mathcal{S}$, respectively. Also, $(\mathbf{r}', \mathbf{k}') = (\mathbf{r}, \mathbf{k})$ if $\mathbf{m}' = \mathbf{m}$. The key generation algorithm, denoted $\mathsf{KEM.CCA.KeyGen}$, randomly selects a secret key sk and computes the corresponding public key pk, and returns both of them. We note that essentially all KEM candidates in the NIST PQ project can be written in the above form or in some similar way.

Algorithm 1 KEM.CCA.Encaps

```
Input: pk
Output: c and s
pick a random $\mathbf{m}$
$(\mathbf{r}, \mathbf{k}) \leftarrow H_1(\mathbf{m}, \mathsf{pk})$
$\mathbf{c} \leftarrow \mathsf{PKE.CPA.Enc}(\mathsf{pk}, \mathbf{m}; \mathbf{r})$
$\mathbf{s} \leftarrow H_2(\mathbf{c}, \mathbf{k})$
Return $(\mathbf{c}, \mathbf{s})$
```

Algorithm 2 KEM.CCA.Decaps

```
Input: sk, pk, c
Output: s'
$\mathbf{m}' \leftarrow \mathsf{PKE.CPA.Dec}(\mathsf{sk}, \mathbf{c})$
$(\mathbf{r}', \mathbf{k}') \leftarrow H_1(\mathbf{m}', \mathsf{pk})$
$\mathbf{c}' \leftarrow \mathsf{PKE.CPA.Enc}(\mathsf{pk}, \mathbf{m}'; \mathbf{r}')$
if $(\mathbf{c}' = \mathbf{c})$ then
    return $\mathbf{s}' \leftarrow H_2(\mathbf{c}, \mathbf{k}')$
else
    return $\mathbf{s}' \leftarrow H_2(\mathbf{c}, \mathsf{sk}_r)$, where $\mathsf{sk}_r$ is a random seed in $\mathsf{sk}$
end if
```

The side-channel attack is now described using calls to an oracle that determines whether, in the PKE.CPA.Dec$(\cdot)$ call, a modified ciphertext decrypts to the same "message" or not. To be a bit more precise, we follow the steps in the public KEM.CCA.Encaps algorithm and record the values of a chosen $\mathbf{m}$, and the corresponding computed $\mathbf{r}$ and ciphertext $\mathbf{c}$. Then we modify the ciphertext to $\mathbf{c}' = \mathbf{c} + \mathbf{d}$, where $\mathbf{d}$ denotes a predetermined modification to the ciphertext. Finally, we require that the oracle can tell us if the modified ciphertext is still decrypted to the same message, i.e., whether $\mathbf{m} = \mathsf{PKE.CPA.Dec}(\mathsf{sk}, \mathbf{c}')$. If so, the oracle returns 0. But if $\mathsf{PKE.CPA.Dec}(\mathsf{sk}, \mathbf{c}')$ returns a different message, the oracle returns 1. Finally, we assume that an oracle output of -1 represents a situation when the oracle cannot decisively give an answer. The high-level construction of the oracle is given in Algorithm Algorithm 3.

The notation $t =$ Side-channel.information[X] means that side-channel information of some kind is collected when executing algorithm X. This information is then analyzed in the $F(t)$ analysis algorithm. In our case we are collecting the time of execution through the number of clock cycles and this is the assumed type of side-channel information. At the end of the paper we discuss and argue for the fact that other types of side-channel information can also be used, for example analysis of power or electromagnetic emanations in case of microcontroller or pure

Algorithm 3 Decryption.Error.In.CPAcall.Oracle

```
Input: $\mathbf{m}$, a ciphertext modification $\mathbf{d}$
Output: $b$ (decryption failure or not)
(r, k) ← $H_1(\mathbf{m}, \mathsf{pk})$
$\mathbf{c} \leftarrow$ PKE.CPA.$Enc(\mathsf{pk}, \mathbf{m}; \mathbf{r})$
$\mathbf{c}' \leftarrow \mathbf{c} + \mathbf{d}$
$t \leftarrow$ Side-channel.information[KEM.CCA.Decaps($\mathbf{c}'$)]
$b \leftarrow F(t)$, where $F(t)$ uses the side.channel information to determine
  whether PKE.CPA.Dec($\mathbf{c}'$) returns $\mathbf{m}$ or not ($b = 0$ means returning $\mathbf{m}$,
  $b = 1$ means not returning $\mathbf{m}$, and $b = -1$ means inconclusive)
return $b$
```

hardware implementations.

The design of $F(t)$ is clearly a key part of the attack. Assuming we have found an oracle that can give us decisive answers for some choices of **m** and ciphertext modifications **d**, the final step is to extract information about the secret key used in the PKE.CPA.Dec algorithm. This part will be highly dependent on the actual scheme considered, but a general summary is given in Algorithm Algorithm 4.

Algorithm 4 Secret key recovery

```
Input: $n_1$
Output: the secret key sk
for $i = 0; i < n_1; i \leftarrow i + 1$ do
    find $(\mathbf{m}_i, \mathbf{d}_i)$ such that Decryption.Error.In.CPAcall.Oracle($\mathbf{m}_i, \mathbf{d}_i$)$\in$
      $\{0, 1\}$
end for
Use the determined set $\{((\mathbf{m}_i, \mathbf{d}_i), 0 \le i < n)\}$ to extract the secret key, by
  exploring the relation between the secret key and modifications that cause
  decryption errors in PKE.CPA.Dec.
return sk
```

3.1 Designing the oracle for LWE-based schemes

The main question is how to find **m** and ciphertext modifications **d** such that the measured timing information may reveal whether PKE.CPA.Dec($\mathbf{c}+\mathbf{d}$) inside the KEM.CCA.Decaps($\mathbf{c} + \mathbf{d}$) is returning the same message **m** or not. The general idea is the following.

The side-channel information $t \leftarrow$ timing.information[X] is simply the time (clock cycles) it takes to execute X. The ciphertext of an LWE-based scheme, created in PKE.CPA.Enc(pk, $\mathbf{m}$; $\mathbf{r}$) may consist of several parts, but at a general level we may describe it as

$$\mathbf{c} = g(\mathsf{pk}, \mathbf{m}; \mathbf{r}) + e(\mathbf{r}),$$

where $e(\mathbf{r})$ is a vector of small error values, and $g(\mathsf{pk}, \mathbf{m}; \mathbf{r})$ represents the remaining part of the ciphertext generation in the scheme. Unique for post-quantum schemes is the property that the error vector $e(\mathbf{r})$ may vary a bit without affecting the ability to decrypt the ciphertext to the correct message. So if we introduce a modified ciphertext $\mathbf{c}' = \mathbf{c}+\mathbf{d}$, then the new ciphertext $\mathbf{c}' = g(\mathsf{pk}, \mathbf{m}; \mathbf{r})+e(\mathbf{r})+\mathbf{d}$. Two things can then happen. Either the modification $\mathbf{d}$ is small enough so that it does not cause an error in decryption and $\mathbf{m} \leftarrow \mathsf{PKE.CPA.Dec}(\mathsf{sk}, \mathbf{c}')$; or the modification $\mathbf{d}$ is big enough to cause an error in decryption and $\mathbf{m} \neq \mathbf{m}' \leftarrow \mathsf{PKE.CPA.Dec}(\mathsf{sk}, \mathbf{c}')$;

An observation is that when we have an error in decryption of $\mathbf{c}'$, receiving $\mathbf{m}'$ ($\neq \mathbf{m}$), then re-encrypting $\mathbf{m}'$ in the decapsulation (line 3 of Algorithm Algorithm 2) results in a completely different ciphertext, which is not at all similar to $\mathbf{c}'$. The attack relies on the fact that the side-channel information can be used to distinguish between the two cases.

The key observation used in the paper is now that if we adopt a ciphertext modification of the form

$$\mathbf{d} = (\underbrace{00\cdots 0}_{n-l}\, d_{n-l}d_{n-l+1}\cdots d_{n-1}),$$

i.e., we only modify the last l entries of the ciphertext, we have the two different cases:

Either there is no decryption error, which leads to $\mathbf{m} = \mathbf{m}' \leftarrow \mathsf{PKE.CPA.Dec}(\mathsf{sk}, \mathbf{c}')$, $(\mathbf{r}, \mathbf{k}) = (\mathbf{r}', \mathbf{k}')$, and $\mathbf{c} = \mathsf{PKE.CPA.Enc}(\mathsf{pk}, \mathbf{m}'; \mathbf{r}')$. So in the check in line 4 of Algorithm 2, ($\mathbf{c}' = \mathbf{c}$), we see that $\mathbf{c}'$ and $\mathbf{c}$ are guaranteed identical except for the last l positions.

If there is a decryption error, on the other hand, i.e., $\mathbf{m} \neq \mathbf{m}' \leftarrow \mathsf{PKE.CPA.Dec}(\mathsf{sk}, \mathbf{c}')$, then the next step in the decapsulation, $(\mathbf{r}', \mathbf{k}') \leftarrow H_1(\mathbf{m}', \mathsf{pk})$ leads to completely different values of $(\mathbf{r}', \mathbf{k}')$, which in turn will give a completely different ciphertext $\mathbf{c}$. In particular $\mathbf{c}'$ and $\mathbf{c}$ will most likely have different values already in the beginning of the vectors.

Finally, how can we separate the two cases using timing information? This is possible since the check in line 4 of Algorithm 2, ($\mathbf{c}' = \mathbf{c}$), involves checking the equality of two long vectors and a standard implementation would terminate after finding a position for which equality does not hold. In the first case above, the first $n - l$ positions are equal and we would have to run through and check all of them before we terminate. In the second case, however, it is very likely to terminate the check very quickly. Typical instructions for which this assumption is true is the use of the memcmp function in C or the java.util.Arrays.equals function in Java. The analysis function $F(t)$, in its simplest form, is assumed to have made some initial measurements to establish intervals $\mathcal{I}_0, \mathcal{I}_1$ for the time of execution in the two cases and returns $F(t) = 0$ if the number of clock cycles is in $\mathcal{I}_0$, $F(t) = 1$ if it is in $\mathcal{I}_1$ and $F(t) = -1$ otherwise. In practice, timing measurements are much more complicated and more advanced methods to build $F(t)$ should be considered.

4 The FrodoKEM design and implementation

In the next section, we will apply our general attack on the FrodoKEM scheme, a main candidate of round 2 in the NIST PQ project. FrodoKEM is a lattice-based KEM with security based on the standard LWE problem. It is a conservative design with security proofs. In the document [Nae+19] submitted to NIST, it is claimed that *"All our implementations avoid the use of secret address accesses and secret branches and, hence, are protected against timing and cache attacks."* An implementation of FrodoKEM that can be attacked also appears in Open Quantum Safe [OQS20].

4.1 The FrodoKEM design

FrodoKEM was firstly published in [Bos+16]. We describe the different algorithms in FrodoKEM (from [Nae+19]) for the key generation, the key encapsulation, and the key decapsulation in Algorithm Algorithm 5–Algorithm 7. We refer to the design document [Nae+19] for all the design details and provide only algorithmic descriptions of the relevant parts in the design. We now also use the notation from the design paper.

Briefly, from an initial seed the key generation $\mathsf{FrodoKEM}$.KeyGen generates the secret and public keys. Note that from $\mathsf{pk} = (\mathsf{seed}_{\mathbf{A}}, \mathbf{B})$, we generate $\mathbf{A} = \mathsf{Frodo}.\mathrm{Gen}(\mathsf{seed}_{\mathbf{A}})$. We have the following equation for a key pair $(\mathsf{pk}, \mathsf{sk})$,

$$\mathbf{B} = \mathbf{AS} + \mathbf{E}, \tag{1}$$

where **B**, **E**, $\mathbf{S} \in \mathbb{Z}_q^{n\times\bar{n}}$ and $\mathbf{A} \in \mathbb{Z}_q^{n\times n}$. Note that while **A** and **B** are publicly known both **S** and **E** are secrets and **S** is saved as part of sk to be used in the decapsulation process, later. Here **E**, **S** are error matrices and by this we mean that the entries in the matrices are small values (compared to $\mathbb{Z}_q$) and distributed according to some predetermined distribution χ.

In an encapsulation, a uniformly random key $\mu \xleftarrow{\$} U(\{0,1\}^{\mathsf{len}_\mu})$ is first chosen. It is then used to generate a pseudorandom bit string that in turn determines error matrices $\mathbf{S}', \mathbf{E}', \mathbf{E}''$. A ciphertext now contains two parts, one being $\mathbf{S}'\mathbf{A} + \mathbf{E}'$ and the second part being $\mathbf{S}'\mathbf{B} + \mathbf{E}'' + \mathsf{Frodo}.\mathrm{Encode}(\mu)$. These matrices are converted to bitstrings using the Frodo.Pack and Frodo.UnPack algorithms. The shared key **ss** is computed using the pseudorandomness generator SHAKE.

Algorithm 5 FrodoKEM.KeyGen

```
Input: None.
Output: Key pair $(\mathsf{pk}, \mathsf{sk}')$ with $\mathsf{pk} \in \{0,1\}^{\mathsf{len}_{\mathsf{seed}_\mathbf{A}} + D \cdot n \cdot \bar{n}}$,
    $\mathsf{sk}' \in \{0,1\}^{\mathsf{len}_\mathsf{s} + \mathsf{len}_{\mathsf{seed}_\mathbf{A}} + D \cdot n \cdot \bar{n}} \times \mathbb{Z}_q^{n \times \bar{n}} \times \{0,1\}^{\mathsf{len}_\mathsf{pkh}}$.
Choose uniformly random seeds $\mathsf{s}||\mathsf{seed}_\mathbf{SE}||\mathbf{z} \xleftarrow{\$} U(\{0,1\}^{\mathsf{len}_\mathsf{s} + \mathsf{len}_{\mathsf{seed}_\mathbf{SE}} + \mathsf{len}_\mathsf{z}})$
Generate pseudorandom seed $\mathsf{seed}_\mathbf{A} \leftarrow \mathrm{SHAKE}(\mathbf{z}, \mathsf{len}_{\mathsf{seed}_\mathbf{A}})$
Generate the matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ via $\mathbf{A} \leftarrow \mathsf{Frodo.Gen}(\mathsf{seed}_\mathbf{A})$
Generate pseudorandom bit string
    $(\mathbf{r}^{(0)}, \ldots, \mathbf{r}^{(2n\bar{n}-1)}) \leftarrow \mathrm{SHAKE}(\mathsf{0x5F}||\mathsf{seed}_\mathbf{SE}, 2n\bar{n} \cdot \mathsf{len}_\chi)$
Sample error matrix
    $\mathbf{S} \leftarrow \mathsf{Frodo}.\mathrm{SampleMatrix}((\mathbf{r}^{(0)}, \ldots, \mathbf{r}^{(n\bar{n}-1)}), n, \bar{n}, T_\chi)$
Sample error matrix
    $\mathbf{E} \leftarrow \mathsf{Frodo}.\mathrm{SampleMatrix}((\mathbf{r}^{(n\bar{n})}, \ldots, \mathbf{r}^{(2n\bar{n}-1)}), n, \bar{n}, T_\chi)$
Compute $\mathbf{B} \leftarrow \mathbf{AS} + \mathbf{E}$
Compute $\mathbf{b} \leftarrow \mathsf{Frodo.Pack}(\mathbf{B})$
Compute $\mathsf{pkh} \leftarrow \mathrm{SHAKE}(\mathsf{seed}_\mathbf{A}||\mathbf{b}, \mathsf{len}_\mathsf{pkh})$
return public key $\mathsf{pk} \leftarrow \mathsf{seed}_\mathbf{A}||\mathbf{b}$ and secret key
    $\mathsf{sk}' \leftarrow (\mathsf{s}||\mathsf{seed}_\mathbf{A}||\mathbf{b}, \mathsf{S}, \mathsf{pkh})$
```

In decapsulation, the step $\mathbf{M} \leftarrow \mathbf{C} - \mathbf{B}'\mathsf{S}$ actually computes $\mathsf{Frodo.Encode}(\mu') + \mathbf{S}'\mathbf{E} - \mathbf{E}'\mathbf{S} + \mathbf{E}''$. Since $\mathbf{S}, \mathbf{S}', \mathbf{E}, \mathbf{E}', \mathbf{E}''$ all have small entries, also $\mathbf{S}'\mathbf{E} - \mathbf{E}'\mathbf{S} + \mathbf{E}''$ will have somewhat small entries and is regarded as noise. The Frodo.Decode algorithm removes this noise and returns the initial seed μ'. The decapsulation then continues by re-encrypting using this seed to get the corresponding ciphertext $\mathbf{B}''||\mathbf{C}'$. In line 16 the two ciphertexts are compared to check equality. If they are equal, the correct shared key **ss** is returned.

The Frodo.SampleMatrix algorithm constructs the matrices with small values from a distribution described by a table T_χ, as given in algorithms Algorithm 8 and Algorithm 9.

Algorithms Algorithm 10 and Algorithm 11 gives the encoding and decoding procedures.

Finally, Frodo designs packing and unpacking algorithms to transform matrices with entries in $\mathbb{Z}_q$ to bit strings and vice versa, as described in Algorithm Algorithm 12 and Algorithm Algorithm 13.

The security parameters of FrodoKEM are listed in Table Table 1.

4.2 A useful observation

A useful observation is that Line 16 in Frodo.Decaps (i.e., Algorithm Algorithm 7) is, in the reference implementation, implemented in a standard way using the

Algorithm 6 FrodoKem.Encaps

```
Input: Public Key $\mathsf{pk} = \mathsf{seed}_{\mathbf{A}}||\mathbf{b} \in \{0,1\}^{\mathsf{len}_{\mathsf{seed}_{\mathbf{A}}}+D\cdot n\cdot\bar{n}}$.
Output: Ciphertext $\mathbf{c}_1||\mathbf{c}_2 \in \{0,1\}^{(\bar{m}\cdot n+\bar{m}\cdot\bar{n})D}$ and shared secret
    $\mathbf{ss} \in \{0,1\}^{\mathsf{len}_{\mathbf{ss}}}$.
Choose a uniformly random key $\mu \xleftarrow{\$} U(\{0,1\}^{\mathsf{len}_\mu})$
Compute $\mathsf{pkh} \leftarrow \mathsf{SHAKE}(\mathsf{pk}, \mathsf{len}_{\mathsf{pkh}})$
Generate pseudorandom values
    $\mathsf{seed}_{\mathbf{SE}}||\mathbf{k} \leftarrow \mathsf{SHAKE}(\mathsf{pkh}||\mu, \mathsf{len}_{\mathsf{seed}_{\mathbf{SE}}} + \mathsf{len}_{\mathbf{k}})$
Generate pseudorandom bit string
    $(\mathbf{r}^{(0)}, \ldots, \mathbf{r}^{(2\bar{m}n+\bar{m}\bar{n}-1)}) \leftarrow \mathsf{SHAKE}(\mathsf{0x96}||\mathsf{seed}_{\mathbf{SE}}, (2\bar{m}n+\bar{m}\bar{n})\cdot\mathsf{len}_\chi)$
Sample error matrix
    $\mathbf{S}' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(0)}, \ldots, \mathbf{r}^{(\bar{m}n-1)}), \bar{m}, n, T_\chi)$
Sample error matrix
    $\mathbf{E}' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(\bar{m}n)}, \ldots, \mathbf{r}^{(2\bar{m}n-1)}), \bar{m}, n, T_\chi)$
Generate $\mathbf{A} \leftarrow \mathsf{Frodo.Gen}(\mathsf{seed}_{\mathbf{A}})$
Compute $\mathbf{B}' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}'$
Compute $\mathbf{c}_1 \leftarrow \mathsf{Frodo.Pack}(\mathbf{B}')$
Sample error matrix
    $\mathbf{E}'' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(2\bar{m}n)}, \ldots, \mathbf{r}^{(2\bar{m}n+\bar{m}\bar{n}-1)}), \bar{m}, \bar{n}, T_\chi)$
Compute $\mathbf{B} \leftarrow \mathsf{Frodo.UnPack}(\mathbf{b}, n, \bar{n})$
Compute $\mathbf{V} \leftarrow \mathbf{S}'\mathbf{B} + \mathbf{E}''$
Compute $\mathbf{C} \leftarrow \mathbf{V} + \mathsf{Frodo.Encode}(\mu)$
Compute $\mathbf{c}_2 \leftarrow \mathsf{Frodo.Pack}(\mathbf{C})$
Compute $\mathbf{ss} \leftarrow \mathsf{SHAKE}(\mathbf{c}_1||\mathbf{c}_2||\mathbf{k}, \mathsf{len}_{\mathbf{ss}})$
return ciphertext $\mathbf{c}_1||\mathbf{c}_2$ and shared secret $\mathbf{ss}$
```

following code block.

```
// Is (Bp == BBp & C == CC) = true
if (memcmp(Bp, BBp, 2*PARAMS_N*PARAMS_NBAR) == 0 && memcmp(C,
    CC, 2*PARAMS_NBAR*PARAMS_NBAR) == 0) {
    memcpy(Fin_k, kprime, CRYPTO_BYTES);
} else {
    memcpy(Fin_k, sk_s, CRYPTO_BYTES);
}
```

We follow the attack strategy from the previous section and assume that the attacker modifies the $\mathbf{c}_2$ part in the ciphertext. If the modification does not affect the output of Frodo.Decode, the re-encryption procedure will generate the same tuple $(\mathbf{S}', \mathbf{E}', \mathbf{E}'')$ and the check

```
memcmp(Bp, BBp, 2*PARAMS_N*PARAMS_NBAR) == 0
```

will be satisfied. Thus,

Table 1: Proposed parameters in FrodoKEM.

	n	q	σ	**support of** χ	B	$\bar{m} \times \bar{n}$	Security
Frodo-640	640	2^{15}	2.8	$[-12 \ldots 12]$	2	8×8	1
Frodo-976	976	2^{16}	2.3	$[-10 \ldots 10]$	3	8×8	3
Frodo-1344	1344	2^{16}	1.4	$[-6 \ldots 6]$	4	8×8	5

```
memcmp(C, CC, 2*PARAMS_NBAR*PARAMS_NBAR) == 0
```

will be further executed. On the other hand, if Frodo.Decode outputs a different message μ', the first check will fail and the second check after the && operation will be ignored. This type of mechanics is referred to as Short Circuit Evaluation, and should not be employed to handle sensitive data. This could lead to a significant difference when comparing the executed time.

More importantly though, the function memcmp is not implemented in a constant time manner, meaning that if we change only the last part of **C** it will lead to a longer execution time. We further explore this feature by only changing the last part of **C** to enlarge the timing gap.

5 The attack applied on FrodoKEM

We first mention the adversary model, which is a CCA attack model with timing leakage. In this model, the adversary $\mathcal{A}$ sends a series of (valid or invalid) ciphertexts to the decapsulation oracle $\mathcal{O}$ and could obtain the corresponding decapsulation time information. He then performs further analysis to recover the secret key **S**.

5.1 The details of the attack

With a call to the PKE decryption function Frodo.Decode, FrodoKEM.Decaps computes

$$\mathbf{M} = \mathbf{C} - \mathbf{B}'\mathbf{S} = \text{Frodo.Encode}(\mu) + \mathbf{S}'\mathbf{E} - \mathbf{E}'\mathbf{S} + \mathbf{E}''.$$

The next lemma from [Nae+19] states the error size that can be handled by the Frodo decode algorithm Frodo.Decode.

Lemma 5.1. *Let* $q = 2^D$*,* $B \leq D$*. Then* $\mathsf{dc}(\mathsf{ec}(k) + e) = k$ *for any* $k, e \in \mathbb{Z}$*, such that* $0 \leq k \leq 2^B$ *and* $-q/2^{B+1} \leq e < q/2^{B+1}$*. Here* dc *is the decoding function and* ec *is the encoding function.*

We start by generating a valid ciphertext $(\mathbf{c}_1||\mathbf{c}_2)$, which will be successfully decrypted. This event happens with probability close to one since the designed decryption failure probability of the CCA version of Frodo is very low. Let $\mathbf{E}'''$ denote the noise matrix, i.e.,

$$\mathbf{E}''' = \mathbf{S}'\mathbf{E} - \mathbf{E}'\mathbf{S} + \mathbf{E}''. \tag{2}$$

Note that $\mathbf{S}', \mathbf{E}', \mathbf{E}''$ are known values and $\mathbf{E} = \mathbf{B} - \mathbf{AS}$ due to Eq. (1). If we can determine $\mathbf{E}'''$, we will have linear equations in the secret key value $\mathbf{S}$. We know that all the $\bar{m} \times \bar{n}$ entries in the matrix $\mathbf{E}'''$ belong to the interval $[-q/2^{B+1}, q/2^{B+1}) = [-2^{D-B-1}, 2^{D-B-1})$ because the decryption succeed.

We now show how to recover $\mathbf{E}'''_{i,j}$, the element of the i-th row and j-th column of $\mathbf{E}'''$. We first unpack $\mathbf{c}_2$ to $\mathbf{C}$ by applying Frodo.UnPack$(\mathbf{c}_2)$ and our goal is to decide the value x_0 such that

$$\mathbf{E}'''_{i,j} + x_0 = 2^{D-B-1}.$$

If we add a positive value x to the element of the i-th row and j-th column of $\mathbf{C}$ to form $\mathbf{C}'$, then this operation is equivalent to adding x to $\mathbf{E}'''_{i,j}$. We pack $\mathbf{C}'$ to $\mathbf{c}_2'$ and send the new ciphertext $(\mathbf{c}_1||\mathbf{c}_2')$ to the decapsulation procedure. If we detect a fast execution, we know that a decryption failure occurred and the value $\mathbf{E}'''_{i,j} + x$ should be outside the interval $[-2^{D-B-1}, 2^{D-B-1})$. Since x is picked to be positive, then we know that

$$\mathbf{E}'''_{i,j} + x \geq 2^{D-B-1}.$$

Otherwise, for a slow execution, we know that

$$\mathbf{E}'''_{i,j} + x < 2^{D-B-1}.$$

Since it will definitely lead to a decryption failure if choosing $x = 2^{D-B}$, we could start the binary search by setting the initial interval as $[0, 2^{D-B}]$ and determine x_0 by $(D - B - 1)$ different choices[1] of x.

Due to the implementation of the memcmp function, we intend to introduce the added noise at the tail part of $\mathbf{c}_2$, to enlarge the time difference. Therefore, we aim to recover $\mathbf{E}'''_{\bar{m}-1,j}$, where $0 \leq j < \bar{n}$, for one valid ciphertext $(\mathbf{c}_1||\mathbf{c}_2)$. For such $\bar{n}$ entries, the changes in the ciphertext are limited to the last $\bar{n}$ positions. Thus, if a decryption error is triggered and the re-encrypted ciphertext is a totally different one, the timing difference could be large.

Let N denote the number of valid ciphertexts generated. One pair of generated valid ciphertexts could provide us $\bar{m} \times \bar{n}$ linear equations. For the Frodo parameters, we always have $\bar{m} = \bar{n} = 8$. As described before, we only select $\bar{n}$ equations corresponding to the last $\bar{n}$ entries in $\mathbf{E}'''$ with the largest time difference. Since we

[1] Due to the distribution of $\mathbf{E}'''_{i,j}$ a minor optimization is possible; The binary search midpoint selection is can be skewed towards the more likely values closer to the middle of the range. This makes a small reduction in the average number of necessary binary search evaluations.

have $n \times \bar{n}$ unknown entries in **S**, we need roughly $N \approx n$ valid ciphertexts for a full key-recovery if all the collected linear equations are independent[2]. Then, the complexity can be roughly estimated as $N \times \bar{n} \times (D - B - 1) \times N_{\mathsf{dis}}$, where N_{dis} is the required number of decryption attempts to decide if it is a fast execution or not.

Last, we point out that if errors occur in the process of recovering the value of x_0, one could use a post-processing step like lattice reduction algorithms to handle these errors and to fully recover the secret key. In this case, it would be helpful to reduce the post-processing complexity if a few more equations are collected.

A summary of the attack procedure against FrodoKEM is given in Algorithm Algorithm 14.

5.2 Simulation method

To increase the chances of successfully distinguishing the two outcomes for each step of the binary search algorithm, the following actions were taken to minimize the noise in our experiment and improving the accuracy of the measurements.

- Hyper Threading was turned off in BIOS.
- Intel SpeedStep was turned off in BIOS.
- Linux kernel's scheduling governor was set to 'performance'.
- All unnecessary processes in the system were turned off for the duration of the measurements.
- The measurement program's affinity was set to a single core.
- The remaining system processes' CPU core affinity were set to the other remaining cores.
- The measurement program's priority was set to the highest value.
- The `rdtscp` instruction were used for measurements. This is a serializing version of the `rdtsc` instruction which forces every preceding instruction to complete before allowing the program to continue. This prevents the CPU out-of-order execution from interfering with the measurements.
- Before starting the timer the decapsulation function is executed once, without being measured, to warm up the data and instruction caches.

[2] As q is a large integer, the probability for a matrix to be full-rank is high. One could also collect slightly more than n ciphertexts to ensure that a full-rank matrix will be obtained.

Despite the actions listed above the noise in the measurements are considerable, and critically, the amount of noise and the shape of the histogram seems to be non-constant. We compensate both by increasing the number of samples and also by attempting a reliability estimation of each set of measurements and discard if they do not seem to match what we expect. The rest of this section will be dedicated to explaining how this has been done in the experiment.

Before running the binary search a warmup-phase is executed which ensures that the CPU frequency stabilizes, branch prediction buffers are populated and the cache is filled. These measurements are also used to calculate a very rough “cutoff” limit above which no timing values will be recorded, as they are deemed too noisy to be of any interest.

We begin by observing that the most significant measurements are those which are closest to the minimum, since these are the values least affected by noise. In our experiments, the most effective strategy to distinguish the two distributions was to simply count the number of measurements whose values are lower than a certain small threshold.

We establish a good threshold by profiling with a high number of iterations I_p in 2 stages.

First we generate a set of measurements M_{low}, with $|M_{\text{low}}| = I_p$, as the first part of the profiling step by repeatedly measuring with a single ciphertext modified by a low amount ($x = 1$). The subset $T_{\text{low}} \subset M_{\text{low}}$ is the fraction F_{low} of the values in M_{low} whose measurements are smallest, i.e. $|T_{\text{low}}| = |M_{\text{low}}| * F_{\text{low}}$. F_{low} is a fixed value in the interval $(0..1)$ and has been determined by experimentation (see Section Section 5.3).

$L_{\text{low}} = \max(T_{\text{low}})$ is used to determine the similar fraction F_{high} of values from the second profiling stage M_{high}, whose values were generated by a high amount of modification ($x = 2^{D-B}$). That is to say

$$T_{\text{high}} = \{t | t \in M_{\text{high}}, t \leq L_{\text{low}}\}$$

and

$$F_{\text{high}} = \frac{|T_{\text{high}}|}{|M_{\text{high}}|}.$$

F_{low} (fixed) and F_{high} (dynamic) are used in the next measurement phase where the binary search algorithm decides whether or not it is experiencing a “fast” or “slow” execution for the particular modification x under evaluation.

We use the set of measurements M_x (where $|M_x| = I_m$) to denote the measurements for a certain value of x and T_x the subset of measurements whose values are lower than L_{low}, so

$$T_{\text{x}} = \{t | t \in M_{\text{x}}, t \leq L_{\text{low}}\}.$$

If

$$F_x = \frac{|T_x|}{|M_x|}$$

is closer to F_{high} than to F_{low} then we assume $\mathbf{E}'''_{i,j} + x \geq 2^{D-B-1}$. Likewise if F_x is closer to F_{low} than to F_{high} then we assume $\mathbf{E}'''_{i,j} + x < 2^{D-B-1}$.

Reliability estimation

As previously mentioned, the measurement noise is considerable, due to the total run-time of the decapsulation routine being so large relative to the difference we wish to measure. The probability of making the wrong decision in each step of the binary search algorithm is non-negligible and therefore some additional checks are added, as detailed below.

If

$$F_{\text{low}} + \frac{\Delta F}{4} \leq F_x \leq F_{\text{high}} - \frac{\Delta F}{4},$$

where $\Delta F = F_{\text{high}} - F_{\text{low}}$, then we deem F_x as too uncertain for us to draw any conclusions. In such a case we do another round of measurements until either F_x move beyond one of the limits or we give up. In the latter case we restart the profiling phase and start over for that particular set of indexes (i, j).

Furthermore we additionally redo the binary search steps when they a) have not changed direction[3] in a number of steps or b) when we have narrowed the possible range down to a single value and we wish to confirm our findings. For case a) this helps with detection and recovery of bad early decisions. Case b) is a way to lower the probability of finding an erroneous value due to a bad decision later in the binary search.

Lastly we make sure $F_x \leq F_{\text{high}} + \Delta F$, otherwise we discard the measurements since they indicates that the profile is no longer valid. In that case we restart with a new profiling phase for the indexes i, j.

5.3 Results

The results documented in this section were generated[4] on a i5–4200U CPU running at 1.6GHz using the FrodoKEM-1344-AES variant as implemented in the Open Quantum Safe software library[5] (`liboqs`) and compiled with default compiler flags.

In Fig. 1 we see that the timing difference is in the order of ≈ 4800 reference clock cycles, as measured on our machine. In contrast, the entire FrodoKEM.Decaps

[3] i.e. if we either continuously lower the upper limit or continuously raise the lower limit for a number of consecutive steps, then we retry the last step to guard against an earlier erroneous decision.

[4] Proof of concept implementation available at: `https://github.com/atneit/open-quantum-safe-attacks`

[5] The latest official reference implementation at `https://github.com/Microsoft/PQCrypto-LWEKE` appear to be identical to the implementation in `liboqs`.

function requires ≈ 12.7M clock cycles, in average, to complete when running on the same machine. Thus we need to distinguish differences in the running time of less than 0.04% of the total run-time of a single decapsulation.

Using the method previously described and with $F_{\text{low}} = 1\%$ (see the L_{low} indication in Fig. 1) we get 85000 measured decapsulations per $\mathbf{E}'''_{i,j}$ value, split between 10000×2 for profiling each index of $\mathbf{E}'''{}_{i,j}$ and 5000 for each step of the binary search and confirmation stage. Factoring in retries of the binary search the average number of decapsulations ends up at ≈ 97000. Using these settings no incorrect values of $\mathbf{E}'''{}_{i,j}$ were obtained after collecting data for > 3000 out of the $1344 \times 8 = 10752$ equations necessary for complete key recovery.

5.4 Summary

For FrodoKEM-1344-AES $\mathbf{E}'''$ is a matrix of size 1344×8 and the attack as implemented requires $97000 \times 1344 \times 8 \approx 2^{30}$ measured decapsulations to complete. With an average runtime of 3.3 positions of $\mathbf{E}'''_{i,j}$ per hour (on the limited hardware described above) we can make a complete key recovery in approximately 136 core-days. This is only taking the data collection phase into account, additional computation for solving the linear equations is considered negligible in comparison.

A strategy to lower the sample complexity would be to improve upon our admittedly simple distinguisher for the two timing distributions. Another source of potentially unnecessary samples is the repetition of the profiling phase for each set of indexes and ciphertexts. It can be argued that a simple timing model could be developed which would allow for a reuse of information from a single or smaller number of profiling steps.

The sample complexity can be even lower if we increase the complexity of the post-processing step using lattice reduction algorithms to deal with any decision errors that would follow a reduced number of measured decapsulations.

Last, the complexity for attacking Frodo-640 and Frodo-976 will be lower due to the smaller size of n. The reason is two-folds; we need to collect less equations and also for a fixed post-process cost (again using lattice reduction techniques), we can handle larger decision errors in the binary search.

6 Discussion on attacking other schemes

The new timing attack could also be applied on the NIST PQC round-2 implementations of LAC [Lu+19], HQC [Agu+19b], BIKE[6] [Ara+19a], Rollo [Ara+19b], and RQC [Agu+19a], where the non constant-time function memcmp or a short circuit evaluation is employed in the implementation of the FO transform to check

[6] The attack discussed using the memcmp function appears to not be applicable to BIKE's implementation in the Open Quantum Safe project nor the latest reference implementation (available on `https://bikesuite.org`).

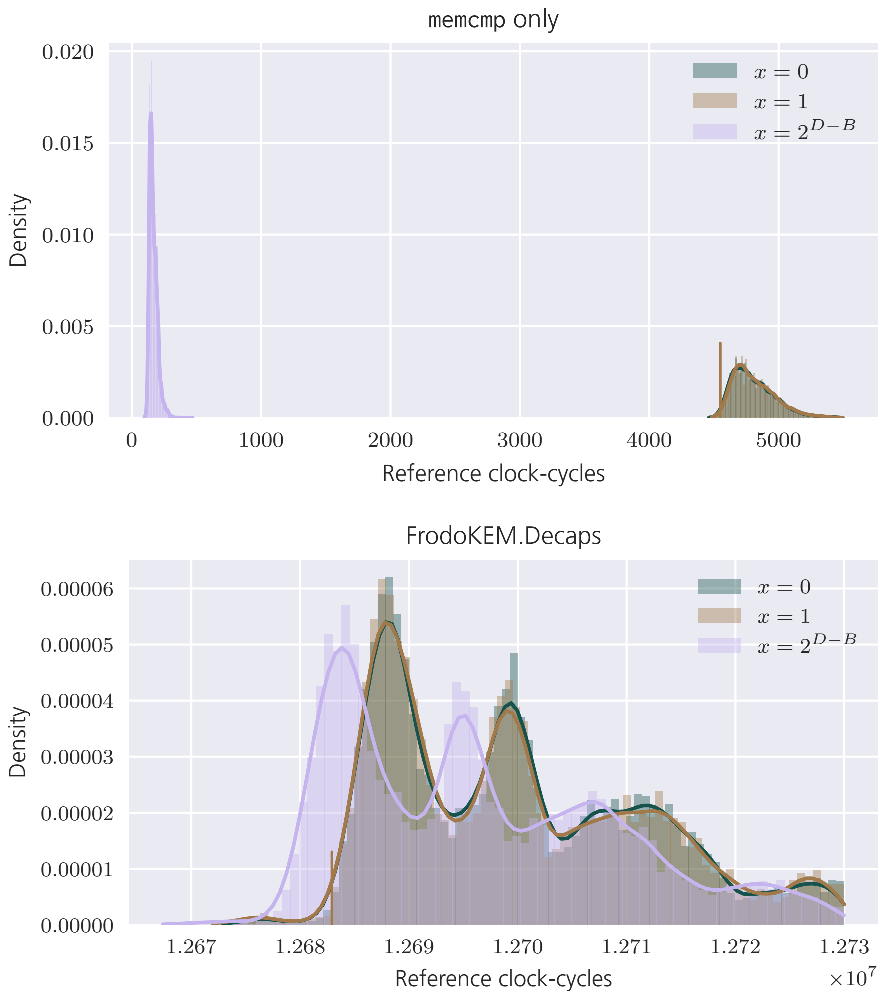


Figure 1: Histograms of timing measurements of only the memcmp function-call (the $\mathbf{C} = \mathbf{C}'$ check) and the entire decapsulation function, respectively. The same ciphertext was sent to the decapsulation function modified in the last position ($i = j = \bar{n} - 1$) of $\mathbf{C}_{i,j}$ by the amount x according to the legend. The curves are the Kernel Density Estimate over the raw measurements. The vertical bar indicates the L_{low} value where $F_1 = 1\%$. In this graph we see that the cutoff limits are at 5500 and 12730000 respectively, above which no values were recorded. 10000 decapsulations each were measured to generate the two figures.

the re-encrypted ciphertexts. The similar designs indicate that they should be vulnerable to the newly proposed attack and the leaked timing information allows a key recovery.

The attack should be slightly adjusted when being applied to schemes like LAC and HQC where additional error-correcting codes are implemented to further reduce the decryption failure probability. In their published implementations, efforts have been made to ensure the BCH decoding to be constant-time in LAC and the recently revised HQC implementation, but a constant-time implementation for the FO transform do not appear to be considered. This knowledge-gap could lead to severe security issues. We refer the interested readers to the appendix for more details on a proposed adaptation of the attack for LAC. The attack on HQC is similar.

We also noted a similar problem in a java implementation of NTRUEncrypt in the NTRU Open Source Project [NTR20], using a non constant-time comparison `java.util.Arrays.equals` for implementing the FO transform.

For all of the schemes mentioned in this paper we suggest to use the constant-time counterpart to `memcmp` (or similar). To do so should not impact the performance of the schemes in any way.

7 Conclusions and future works

We have presented a novel timing attack that can be applied to lattice-based or code-based schemes that use the FO transformation. It uses timing leakage in the ciphertext comparison step of the FO transformation and it can potentially recover the secret key. We applied it on FrodoKEM and implemented the attack with the result that we, with experiments, extrapolated that enough information to determine the secret key can be obtained by measuring about 2^{30} decapsulation calls. Additionally we derived some details of how the attack can be adapted to work on LAC, see appendix.

The attack applies also a number of other round 2 candidates, although we did not fully derive the details of the attack for other schemes, nor did we implement the attack on them.

Note that the current attack could not directly be applied to the submitted reference implementations of, for example, NewHope [Pop+19], Kyber [Sch+19], classic McEliece [Ber+19], or the latest implementation of BIKE (including the BIKE implementation in Open Quantum Safe).

Following the basic idea of the attack on FrodoKEM, one can note that the bitwise sum of the two ciphertexts to be compared have quite different Hamming weights in the two cases of generating a decryption failure or not in the call to the CPA-secure primitive. If a modified ciphertext is decrypted to the same message, the Hamming weights of the xor differences is very low. Such a scenario opens up for other types of side-channel attacks like power analysis, since operations on binary data with different Hamming weight is a typical source of leakage in power analysis.

8 Acknowledgements

The authors would like to thank the anonymous reviewers from CRYPTO 2020 for their helpful comments. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, by the Norwegian Research Council (Grant No. 247742/070), by the SSF SURPRISE project and the Swedish Research Council (Grant No. 2019-04166).

References

[Agu+19a] C. Aguilar Melchor, N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, G. Zémor, et al. *RQC*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`. National Institute of Standards and Technology, 2019.

[Agu+19b] C. Aguilar Melchor, N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, E. Persichetti, et al. *HQC*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`. National Institute of Standards and Technology, 2019.

[Ara+19a] N. Aragon, P. Barreto, et al. *BIKE*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`. National Institute of Standards and Technology, 2019.

[Ara+19b] N. Aragon, O. Blazy, et al. *ROLLO*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`. National Institute of Standards and Technology, 2019.

[Bae+19] C. Baetu, F. B. Durak, L. Huguenin-Dumittan, A. Talayhan, and S. Vaudenay. "Misuse Attacks on Post-quantum Cryptosystems". In: *Advances in Cryptology – EUROCRYPT 2019, Part II*. Ed. by Y. Ishai and V. Rijmen. Vol. 11477. Lecture Notes in Computer Science. Darmstadt, Germany: Springer, Heidelberg, Germany, May 2019, pp. 747–776.

[Bau+19] A. Bauer, H. Gilbert, G. Renault, and M. Rossi. "Assessment of the Key-Reuse Resilience of NewHope". In: *Topics in Cryptology – CT-RSA 2019*. Ed. by M. Matsui. Vol. 11405. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, Mar. 2019, pp. 272–292.

[BB03] D. Brumley and D. Boneh. "Remote Timing Attacks Are Practical". In: *USENIX Security 2003: 12th USENIX Security Symposium*. Washington, DC, USA: USENIX Association, Aug. 2003.

[Ber+17] D. J. Bernstein, L. G. Bruinderink, T. Lange, and L. Panny. *HILA5 Pindakaas: On the CCA security of lattice-based encryption with error correction*. Cryptology ePrint Archive, Report 2017/1214. `https://eprint.iacr.org/2017/1214`. 2017.

[Ber+19] D. J. Bernstein, T. Chou, et al. *Classic McEliece*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`. National Institute of Standards and Technology, 2019.

[Bos+16] J. W. Bos et al. "Frodo: Take off the Ring! Practical, Quantum-Secure Key Exchange from LWE". In: *ACM CCS 2016: 23rd Conference on Computer and Communications Security*. Ed. by E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi. Vienna, Austria: ACM Press, Oct. 2016, pp. 1006–1018.

[Bru+16] L. G. Bruinderink, A. Hülsing, T. Lange, and Y. Yarom. "Flush, Gauss, and Reload - A Cache Attack on the BLISS Lattice-Based Signature Scheme". In: *Cryptographic Hardware and Embedded Systems – CHES 2016*. Ed. by B. Gierlichs and A. Y. Poschmann. Vol. 9813. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2016, pp. 323–345.

[DAn+19a] J.-P. D'Anvers, Q. Guo, T. Johansson, A. Nilsson, F. Vercauteren, and I. Verbauwhede. "Decryption Failure Attacks on IND-CCA Secure Lattice-Based Schemes". In: *PKC 2019: 22nd International Conference on Theory and Practice of Public Key Cryptography, Part II*. Ed. by D. Lin and K. Sako. Vol. 11443. Lecture Notes in Computer Science. Beijing, China: Springer, Heidelberg, Germany, Apr. 2019, pp. 565–598.

[DAn+19b] J.-P. D'Anvers, M. Tiepelt, F. Vercauteren, and I. Verbauwhede. "Timing attacks on error correcting codes in post-quantum schemes". In: *Proceedings of ACM Workshop on Theory of Implementation Security Workshop*. 2019, pp. 2–9.

[Fac+18] A. Facon, S. Guilley, M. Lec'Hvien, A. Schaub, and Y. Souissi. "Detecting cache-timing vulnerabilities in post-quantum cryptography algorithms". In: *2018 IEEE 3rd International Verification and Security Workshop (IVSW)*. IEEE. 2018, pp. 7–12.

[Flu16] S. Fluhrer. *Cryptanalysis of ring-LWE based key exchange with key share reuse*. Cryptology ePrint Archive, Report 2016/085. `http://eprint.iacr.org/2016/085`. 2016.

[FO99] E. Fujisaki and T. Okamoto. "Secure Integration of Asymmetric and Symmetric Encryption Schemes". In: *Advances in Cryptology – CRYPTO'99*. Ed. by M. J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1999, pp. 537–554.

[GJY19] Q. Guo, T. Johansson, and J. Yang. "A Novel CCA Attack Using Decryption Errors Against LAC". In: *Advances in Cryptology – ASIACRYPT 2019, Part I*. Ed. by S. D. Galbraith and S. Moriai. Vol. 11921. Lecture Notes in Computer Science. Kobe, Japan: Springer, Heidelberg, Germany, Dec. 2019, pp. 82–111.

[HHK17] D. Hofheinz, K. Hövelmanns, and E. Kiltz. "A Modular Analysis of the Fujisaki-Okamoto Transformation". In: *TCC 2017: 15th Theory of Cryptography Conference, Part I*. Ed. by Y. Kalai and L. Reyzin. Vol. 10677. Lecture Notes in Computer Science. Baltimore, MD, USA: Springer, Heidelberg, Germany, Nov. 2017, pp. 341–371.

[How+03a] N. Howgrave-Graham, P. Q. Nguyen, et al. "The Impact of Decryption Failures on the Security of NTRU Encryption". In: *Advances in Cryptology – CRYPTO 2003*. Ed. by D. Boneh. Vol. 2729. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2003, pp. 226–246.

[How+03b] N. Howgrave-Graham, J. H. Silverman, A. Singer, and W. Whyte. *NAEP: Provable Security in the Presence of Decryption Failures*. Cryptology ePrint Archive, Report 2003/172. `http://eprint.iacr.org/2003/172`. 2003.

[Koc96] P. C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In: *Advances in Cryptology – CRYPTO'96*. Ed. by N. Koblitz. Vol. 1109. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1996, pp. 104–113.

[Lu+19] X. Lu et al. *LAC*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`. National Institute of Standards and Technology, 2019.

[McE78] R. J. McEliece. "A public-key cryptosystem based on algebraic". In: *Coding Thv* 4244 (1978), pp. 114–116.

[Nae+19] M. Naehrig et al. *FrodoKEM*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`. National Institute of Standards and Technology, 2019.

[NIS18] NIST. *NIST Post-Quantum Cryptography Standardization*. `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization` Accessed: 2018-09-24. 2018.

[NTR20] NTRU. *NTRU Open Source Project*. `https://github.com/NTRUOpenSourceProject` Accessed: 2020-02-10. 2020.

[OQS20] OQS. *Open Quantum Safe*. `https://openquantumsafe.org` Accessed: 2020-01-21. 2020.

[Pop+19] T. Poppelmann et al. *NewHope*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`. National Institute of Standards and Technology, 2019.

[Sch+19] P. Schwabe et al. *CRYSTALS-KYBER*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`. National Institute of Standards and Technology, 2019.

[Sho94] P. W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *35th Annual Symposium on Foundations of Computer Science*. Santa Fe, NM, USA: IEEE Computer Society Press, Nov. 1994, pp. 124–134.

[Sma16] N. P. Smart. *Cryptography Made Simple*. Information Security and Cryptography. Springer, Heidelberg, Germany, 2016.

[Str10] F. Strenzke. "A Timing Attack against the Secret Permutation in the McEliece PKC". In: *The Third International Workshop on Post-Quantum Cryptography, PQCRYPTO 2010*. Ed. by N. Sendrier. Darmstadt, Germany: Springer, Heidelberg, Germany, May 2010, pp. 95–107.

[Str13] F. Strenzke. “Timing Attacks against the Syndrome Inversion in Code-Based Cryptosystems”. In: *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013*. Ed. by P. Gaborit. Limoges, France: Springer, Heidelberg, Germany, June 2013, pp. 217–230.

Appendix A: The attack on LAC

In this section, we focus on applying the new attack on LAC [Lu+19]. Similar procedures can also be used in attacking HQC after some minor modifications, and the framework is the same as we described here for general schemes with ECC.

LAC is a lattice-based proposal to the NIST Post-quantum Standardization project that has advanced to the second round. It includes three different versions, LAC128-v2, LAC192-v2, and LAC256-v2, aiming for the security levels of 128, 192, and 256 bits, respectively. We take LAC128-v2 as an instance to describe how the new timing attacks can be applied to the LAC proposal. The concrete parameters of LAC128-v2 are shown in Table Table 2.

Table 2: Proposed parameters of LAC128-v2.

n	q	$\mathcal{R}$	h	η	Distribution	ecc	bit-er	DFR	Security
512	251	$\frac{\mathbb{Z}_q[x]}{\langle x^n+1\rangle}$	256	400	$\Psi_1, \Psi_1^{n,h}$	BCH$[511, 256, 33]$	$2^{-12.61}$	2^{-116}	I

Notations. Let the modulus be q, and the underlying polynomial ring be $\mathcal{R} = \mathbb{Z}_q/(x^n + 1)$. The distribution Ψ_1 randomly outputs 0 with probability $1/2$ and outputs 1 (or -1) with probability $1/4$. For a positive integer h, the distribution $\Psi_1^{n,h}$ outputs a length-n vector with $h/2$ ones, $h/2$ minus-ones, and $(n - h)$ zeros.

The LAC design. The LAC scheme has an extreme design with a very small q and therefore the position-wise error probability (denoted as bit-er in Table Table 2) is rather large. It uses an error correcting codes (ECC) to further reduce the overall decryption error probability. The concrete code used is a BCH code with length 511, dimension 256, and minimum distance 33. Thus, the error correcting capability of the employed BCH code is 16. This code is a shorten code and the parameter η denotes the size of the information and the redundant data. In the second round submission, the designers employ a compression function to reduce the ciphertext size in transmission.

The algorithms in the LAC proposal for key generation, key encapsulation, and key decapsulation can be found in [Lu+19]. We list them here for completeness.

A general approach for attacking schemes with ECC. We now describe the general attacking framework. Similar to the FrodoKEM, the ciphertext is generally of the form $(\mathbf{c}_1||\mathbf{c}_2)$ and the decoding is done by computing $\mathbf{c}_2 - \mathbf{c}_1\mathbf{s}$, where $\mathbf{s}$ is the secret key. In the schemes with ECC, however, the ambient space is a polynomial ring where a vector can be also treated as a polynomial. Thus, we could mix the use of the notations of $s(x)$ and $\mathbf{s}$ if there is no ambiguity.

The main tool is still to introduce additional noise in the last part of $\mathbf{c}_2$, which can be done by adding a large value to a position in the Euclidean case (for LAC) or by filliping many bits within a small chunk of positions in the Hamming case (for HQC). The aim is then to recover the noise variables w.r.t. certain positions, which are linear functions of the secret key by testing the minimal added noise size that could lead to a decryption error. The decryption will lead to a fast checking in the non constant-time FO implementation since the re-encrypted ciphertext are random vectors leading to a difference at the beginning part of the ciphertexts $\mathbf{c}_1$, as described. However, since the overall decryption error happen only if strictly more than δ_0 position errors occur in the decryption phase, the attack strategy is less straightforward.

Since one could trigger a position error using the described process of introducing a rather large noise, the attacker is capable of adding position errors at the last positions to ensure the number of position errors to be exactly δ_0. The attacker is then capable of detecting if an uncontrolled position is erroneous or error-free — he could add a big noise to that position and this will lead to a decryption error if the position is error-free.

The attacker picks a position close to the controlled error positions that are error-free and tests the error value in that position by the binary search as discussed in the previous section for FrodoKEM. The error term is generally in the form of $w(x) = e(x)r(x) - e_1(x)s(x) + e_2(x)$, where $e(x)$ and $s(x)$ contain the secret key information, and $r(x)$, $e_1(x)$, and $e_2(x)$ could be known from the encapsulation algorithm. Thus, we can obtain one linear equation whose unknowns are the coefficients of $e(x)$ and $s(x)$ from the detected one coefficient (position) of $w(x)$. Also, note that we already know n linear equations w.r.t. the coefficients of $e(x)$ and $s(x)$ from the key generation procedure. The attack proceeds by generating more ciphertexts until a sufficient number of equations are collected for a full key-recovery.

Dealing with the compression function. In the round-2 submission of LAC, a ciphertext compression technique is employed, introducing an additional rounding error. Thus, the general attack approach should be tweaked to handle this unknown noise part.

In the reference implementation of LAC128-v2, the comparison between the ciphertext and the re-encrypted one is implemented as follows.

```
//verify
if(memcmp(c,c_v,CIPHER_LEN)!=0)
{
  //k=hash(hash(sk)|c)
  hash((unsigned char*)sk,DIM_N,buf);
  hash(buf,MESSAGE_LEN+CIPHER_LEN,k);
}
```

Here, $\mathbf{c} = (\mathbf{c}_1||\mathbf{c}_{2,\text{compressed}})$, where $\mathbf{c}_1$ is a length-512 vector (or polynomial) and $\mathbf{c}_{2,\text{compressed}}$ is the compressed ciphertext part of length 200, and $\mathbf{c}_v$ is the re-encrypted ciphertext of the same size. Each byte in $\mathbf{c}_{2,\text{compressed}}$ is the concatenation of the 4 most significant bits in the two corresponding positions in

$\mathbf{c}_2$. Thus, the final noise term should include a new polynomial $e_3(x)$ from the compression operation. Since this polynomial is from a rounding operation and unknown to us, the above general approach can not be directly applied.

On the other hand, it is already shown in [DAn+19a] that if one can detect if a position is erroneous, then a few thousand such erroneous positions could lead to a full recovery. We next show in detail the procedure of determining the erroneous positions, which is an elaboration of the method described in the general attack.

For LAC128-v2, the position-wise decoding is successful if the error variable corresponding to that position lies in the interval of $[-62, 62]$, and in this case, the value has a small absolute value with high probability. Let $\mathbf{c}_2$ be the vector of length 400, which will be compressed to $\mathbf{c}_{2,\mathsf{compressed}}$ of length 200 in the ciphertext. Then, it will cause a position error with high probability if adding 125 to a position in $\mathbf{c}_2$ and compressing the new $\mathbf{c}_2$ to $\mathbf{c}'_{2,\mathsf{compressed}}$ by the compression function. Since the position error probability for LAC128-v2 is only $2^{-12.61}$ (see Table Table 2), it will have $\delta_0 = 16$ position errors with high probability if one adds 125 to the last 16 entries in $\mathbf{c}_2$. The threshold δ_0 is set to be 16 since the error correcting capability of the employed BCH codes is exactly 16. With some probability (of about $384/2^{12.6}$), one could find one position error originally occurs in the first 384 positions of $\mathbf{c}_2$. Thus, it will lead to a different re-encryption if one adds 125 to the last 16 positions of $\mathbf{c}_2$, but not if only 15 positions are changed. After finding this state, the attacker can keep the last 15 positions of $\mathbf{c}_2$ added by 125 and also add the i-th position in $\mathbf{c}_2$ by 125. He then compresses the invalid ciphertext and sends it to the decryption oracle. If the i-th position is already erroneous, the number of position errors will not be increased and a fast check cannot be detected via the timing channel. All additions are operated over $\mathbb{Z}_q$.

The other LAC versions can be attacked in a similar manner, and the attack version on LAC256-v2 with the D2 encoding would need a slight adjustment.

Algorithm 7 FrodoKEM.Decaps

```
Input: Ciphertext $\mathbf{c}_1||\mathbf{c}_2 \in \{0,1\}^{(\bar{m}\cdot n+\bar{m}\cdot\bar{n})D}$, secret
    $\mathsf{sk}' \in \{0,1\}^{\mathsf{len_s}+\mathsf{len_{seed_A}}+D\cdot n\cdot\bar{n}} \times \mathbb{Z}_q^{n\times\bar{n}} \times \{0,1\}^{\mathsf{len_{pkh}}}$.
Output: Shared secret $\mathbf{ss} \in \{0,1\}^{\mathsf{len_{ss}}}$.
$\mathbf{B}' \leftarrow \mathsf{Frodo.UnPack}(\mathbf{c}_1)$
$\mathbf{C} \leftarrow \mathsf{Frodo.UnPack}(\mathbf{c}_2)$
Compute $\mathbf{M} \leftarrow \mathbf{C} - \mathbf{B}'\mathbf{S}$
Compute $\mu' \leftarrow \mathsf{Frodo.Decode}(\mathbf{M})$
Parse $\mathsf{pk} \leftarrow \mathsf{seed}_{\mathbf{A}}||\mathbf{b}$
Generate pseudorandom values
    $\mathsf{seed}_{\mathbf{SE}'}||\mathbf{k}' \leftarrow \mathsf{SHAKE}(\mathsf{pkh}||\mu', \mathsf{len}_{\mathsf{seed}_{\mathbf{SE}}} + \mathsf{len}_{\mathbf{k}})$
Generate pseudorandom bit string
    $(\mathbf{r}^{(0)}, \ldots, \mathbf{r}^{(2\bar{m}n+\bar{m}\bar{n}-1)}) \leftarrow \mathsf{SHAKE}(\mathsf{0x96}||\mathsf{seed}_{\mathbf{SE}'}, (2\bar{m}n + \bar{m}\bar{n}) \cdot \mathsf{len}_\chi)$
Sample error matrix
    $\mathbf{S}' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(0)}, \ldots, \mathbf{r}^{(\bar{m}n-1)}), \bar{m}, n, T_\chi)$
Sample error matrix
    $\mathbf{E}' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(\bar{m}n)}, \ldots, \mathbf{r}^{(2\bar{m}n-1)}), \bar{m}, n, T_\chi)$
Generate $\mathbf{A} \leftarrow \mathsf{Frodo.Gen}(\mathsf{seed}_{\mathbf{A}})$
Compute $\mathbf{B}'' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}'$
Sample error matrix
    $\mathbf{E}'' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(2\bar{m}n)}, \ldots, \mathbf{r}^{(2\bar{m}n+\bar{m}\bar{n}-1)}), \bar{m}, \bar{n}, T_\chi)$
Compute $\mathbf{B} \leftarrow \mathsf{Frodo.UnPack}(\mathbf{b}, n, \bar{n})$
Compute $\mathbf{V} \leftarrow \mathbf{S}'\mathbf{B} + \mathbf{E}''$
Compute $\mathbf{C}' \leftarrow \mathbf{V} + \mathsf{Frodo.Encode}(\mu')$
if $\mathbf{B}'||\mathbf{C} = \mathbf{B}''||\mathbf{C}'$ then
    Return shared secret $\mathbf{ss} \leftarrow \mathsf{SHAKE}(\mathbf{c}_1||\mathbf{c}_2||\mathbf{k}', \mathsf{len}_{\mathbf{ss}})$
else
    Return shared secret $\mathbf{ss} \leftarrow \mathsf{SHAKE}(\mathbf{c}_1||\mathbf{c}_2||\mathsf{s}, \mathsf{len}_{\mathbf{ss}})$
end if
```

Algorithm 8 Frodo.Sample

```
Input: A (random) bit string $\mathbf{r} = (\mathbf{r}_0, \ldots, \mathbf{r}_{\mathsf{len}_\chi - 1}) \in \{0,1\}^{\mathsf{len}_\chi}$, the table
    $T_\chi = (T_\chi(0), \ldots, T_\chi(s))$.
Output: A sample $e \in \mathbb{Z}$.
$t \leftarrow \sum_{i=1}^{\mathsf{len}_\chi - 1} \mathbf{r}_i \cdot 2^{i-1}$
$e \leftarrow 0$
for $z = 0; z < s; z \leftarrow z + 1$ do
    if $t > T_\chi(z)$ then
        $e \leftarrow e + 1$
    end if
end for
$e \leftarrow (-1)^{\mathbf{r}_0} \cdot e$
return $\mathbf{C}$
```

Algorithm 9 Frodo.SampleMatrix

```
Input: A (random) bit string $\mathbf{r} = (\mathbf{r}^{(0)}, \ldots, \mathbf{r}^{(n_1 \times n_2 - 1)}) \in \{0,1\}^{n_1 n_2 \cdot \mathsf{len}_\chi}$, the
    table $T_\chi$.
Output: A sample $\mathbf{E} \in \mathbb{Z}^{n_1 \times n_2}$.
for $i = 0; i < n_1; i \leftarrow i + 1$ do
    for $j = 0; j < n_2; j \leftarrow j + 1$ do
        $\mathbf{E}_{i,j} \leftarrow$ Frodo.Sample($\mathbf{r}^{(i \cdot n_2 + j)}, T_\chi$)
    end for
end for
return $\mathbf{E}$
```

Algorithm 10 Frodo.Encode

```
Input: Bit string $\mathbf{k} \in \{0,1\}^l$, $l = B \cdot \bar{m} \cdot \bar{n}$.
Output: Matrix $\mathbf{K} \in \mathbb{Z}_q^{\bar{m} \times \bar{n}}$.
for $i = 0; i < \bar{m}; i \leftarrow i + 1$ do
    for $j = 0; j < \bar{n}; j \leftarrow j + 1$ do
        $k = \sum_{l=0}^{B-1} \mathbf{k}_{(i \cdot \bar{n} + j)B + l} \cdot 2^l$
        $\mathbf{K}_{i,j} \leftarrow \mathsf{ec}(k) = k \cdot q / 2^B$
    end for
end for
return $\mathbf{K} = (\mathbf{K}_{i,j})_{0 \le i \le \bar{m}, 0 \le j \le \bar{n}}$
```

Algorithm 11 Frodo.Decode

```
Input: Matrix $\mathbf{K} \in \mathbb{Z}_q^{\bar{m}\times\bar{n}}$.
Output: Bit string $\mathbf{k} \in \{0,1\}^l$, $l = B \cdot \bar{m} \cdot \bar{n}$.
for $i = 0; i < \bar{m}; i \leftarrow i + 1$ do
    for $j = 0; j < \bar{n}; j \leftarrow j + 1$ do
        $k \leftarrow \mathsf{dc}(\mathbf{K}_{i,j}) = \lfloor \mathbf{K}_{i,j} \cdot 2^B / q \rceil \mod 2^B$
        $k = \sum_{l=0}^{B-1} k_l \cdot 2^l$ where $k_l \in \{0,1\}$
        for $l = 0; l < D; l \leftarrow l + 1$ do
            $\mathbf{k}_{(i\cdot\bar{n}+j)B+l} \leftarrow k_l$
        end for
    end for
end for
return $\mathbf{k}$
```

Algorithm 12 Frodo.Pack

```
Input: Matrix $\mathbf{C} \in \mathbb{Z}_q^{n_1\times n_2}$
Output: Bit string $\mathbf{b} \in \{0,1\}^{D\cdot n_1\cdot n_2}$
for $i = 0; i < n_1; i \leftarrow i + 1$ do
    for $j = 0; j < n_2; j \leftarrow j + 1$ do
        $\mathbf{C}_{i,j} = \sum_{l=0}^{D-1} c_l \cdot 2^l$ where $c_l \in \{0,1\}$
        for $l = 0; l < D; l \leftarrow l + 1$ do
            $\mathbf{b}_{(i\cdot n_2+j)D+l} \leftarrow c_{D-1-l}$
        end for
    end for
end for
return $\mathbf{b}$
```

Algorithm 13 Frodo.Unpack

```
Input: Bit string $\mathbf{b} \in \{0,1\}^{D\cdot n_1\cdot n_2}$, $n_1$, $n_2$.
Output: Matrix $\mathbf{C} \in \mathbb{Z}_q^{n_1\times n_2}$
for $i = 0; i < n_1; i \leftarrow i + 1$ do
    for $j = 0; j < n_2; j \leftarrow j + 1$ do
        $\mathbf{C}_{i,j} = \sum_{l=0}^{D-1} \mathbf{b}_{(i\cdot n_2+j)D+l} \cdot 2^{D-1-l}$
    end for
end for
return $\mathbf{C}$
```

Algorithm 14 Timing attack on Frodo.KEM

```
Input: The public key $\mathsf{pk} \leftarrow (\mathsf{seed}_{\mathbf{A}}, \mathbf{B})$.
Output: The secret key $\mathbf{S}$.
for $t = 0; t < N; t \leftarrow t + 1$ do
    Generate a valid ciphertext $(\mathbf{c}_1 || \mathbf{c}_2)$
    for $i = 0; i < \bar{n}; i \leftarrow i + 1$ do
        Use the binary search to recover $\mathbf{E}'''_{(\bar{m}-1),i}$
    end for
end for
Recover $\mathbf{S}$ from $\mathbf{E}'''_{(\bar{m}-1),i}$ values by solving linear equations
return S
```

Algorithm 15 LAC.KeyGen()

```
Output: A pair of public key and secret key $(pk, sk)$.
    $\mathsf{seed}_{\mathsf{a}} \xleftarrow{\$} \mathcal{S}$;
    $\mathbf{a} \leftarrow \mathsf{Samp}(U(\mathcal{R}); \mathsf{seed}_{\mathsf{a}}) \in \mathcal{R}$;
    $\mathbf{s} \xleftarrow{\$} \Psi_{\sigma}^{n}$;
    $\mathbf{e} \xleftarrow{\$} \Psi_{\sigma}^{n}$;
    $\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e} \in \mathcal{R}$;
    return $(pk := (\mathsf{seed}_{\mathbf{a}}, \mathbf{b}), sk := \mathbf{s})$;
```

Algorithm 16 LAC.CCA.Enc(pk; $\mathsf{seed}_{\mathbf{m}}$)

```
Output: A ciphertext and encapsulation key pair $(\mathbf{c}, K)$.
    $\mathbf{m} \leftarrow \mathsf{Samp}(U(\mathcal{M}); \mathsf{seed}_{\mathbf{m}}) \in \mathcal{M}$;
    $\mathbf{seed} \leftarrow G(\mathbf{m}) \in \mathcal{S}$;
    $\mathbf{c} \leftarrow \mathsf{LAC.CPA.Enc}(pk, \mathbf{m}; \mathsf{seed})$;
    $K \leftarrow H(\mathbf{m}, \mathbf{c}) \in \{0, 1\}^{l_k}$;
    return $(\mathbf{c}, K)$;
```

Algorithm 17 LAC.CCA.Dec(sk; $\mathbf{c}$)

```
Output: An encapsulation key $(K)$.
    $\mathbf{m} \leftarrow \mathsf{LAC.CPA.Dec}(sk, \mathbf{c})$;
    $K \leftarrow H(\mathbf{m}, \mathbf{c})$;
    $\mathbf{seed} \leftarrow G(\mathbf{m}) \in \mathcal{S}$;
    $\mathbf{c}' \leftarrow \mathsf{LAC.CPA.Enc}(pk, \mathbf{m}; \mathsf{seed})$;
    if $\mathbf{c}' \neq \mathbf{c}$ then
        $K \leftarrow H(H(sk), \mathbf{c})$;
    end if
    return $K$;
```

Algorithm 18 $\mathsf{LAC.CPA.Enc}(pk = (\mathsf{seed}_{\mathbf{a}}, \mathbf{b}), \mathbf{m} \in \mathcal{M}; \mathsf{seed} \in \mathcal{S})$

```
Output: A ciphertext c.
  $\mathbf{a} \leftarrow \mathsf{Samp}(U(\mathcal{R}); \mathsf{seed}_{\mathbf{a}}) \in \mathcal{R}$;
  $\mathbf{c_m} \leftarrow \mathsf{ECCEnc}(\mathbf{m}) \in \{0,1\}^{l_v}$;
  $(\mathbf{r}, \mathbf{e}_1, \mathbf{e}_2) \leftarrow \mathsf{Samp}(\Psi_\sigma^n, \Psi_\sigma^n, \Psi_\sigma^{l_v}; \mathsf{seed})$;
  $\mathbf{c}_1 \leftarrow \mathbf{ar} + \mathbf{e}_1 \in \mathcal{R}$;
  $\mathbf{c}_2 \leftarrow (\mathbf{br})_{l_v} + \mathbf{e}_2 + \lfloor \frac{q}{2} \rfloor \cdot \mathbf{c_m} \in \mathbb{Z}_q^{l_v}$;
  return $\mathbf{c} := (\mathbf{c}_1, \mathbf{c}_2) \in \mathcal{R} \times \mathbb{Z}_q^{l_v}$;
```

Algorithm 19 $\mathsf{LAC.CPA.Dec}(sk = \mathbf{s}; \mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2))$

```
Output: A plaintext m.
  $\mathbf{u} \leftarrow \mathbf{c}_1\mathbf{s} \in \mathcal{R}$;
  $\mathbf{c}'_{\mathbf{m}} \leftarrow \mathbf{c}_2 - (\mathbf{u})_{l_v} \in \mathbb{Z}_q^{l_v}$;
  for $i = 0$ to $l_v - 1$ do
    if $\frac{q}{4} \leq \mathbf{c}'_{\mathbf{m}i} < \frac{3q}{4}$ then
      $\mathbf{c}_{\mathbf{m}i} \leftarrow 1$
    else
      $\mathbf{c}_{\mathbf{m}i} \leftarrow 0$
    end if
  end for
  $\mathbf{m} \leftarrow \mathsf{ECCDec}(\mathbf{c_m})$;
  return $\mathbf{m}$;
```

A Weighted Bit Flipping Decoder for QC-MDPC-based Cryptosystems

Abstract

A new "Weighted Bit-flipping" (WBF) iterative decoder is presented and analyzed with respect to its Decoding Failure Rate (DFR). We show that the DFR is indeed lower than that of the BGF decoder as suggested by the BIKE third round submission to the NIST PQC standardization process. The WBF decoder requires more iterations to complete than BGF, but by creating a hybrid decoder we show that a lower DFR compared to that of the BGF decoder can still be achieved while keeping the computational tradeoff to a minimum.

1 Introduction

It is well-known that quantum computers (QC) can break the ubiquitous public key cryptosystems (PKC) RSA and ECC as well as the key-agreement protocols DH and ECDH. This is due to QCs ability to solve the integer factorization and the discrete logarithm problems in polynomial time. To prepare against the development of QCs of sufficient scale to threaten online communications, many potential replacement algorithms have been suggested [McE78; Ara+17; Bos+17; Mel+18] in the field of post-quantum cryptography (PQC). The code-based PKC scheme McEliece [McE78] is one of these proposed replacements. The security of this PKC is based on hardness of decoding a random linear code (from the field of coding theory). However, relatively recently, new versions this code-based PKC were proposed.

A. Nilsson, I. E. Bocharova, B. D. Kudryashov, and T. Johansson. "A Weighted Bit Flipping Decoder for QC-MDPC-based Cryptosystems". In: *2021 IEEE International Symposium on Information Theory (ISIT)*. 2021, pp. 1266–1271

One of the more promising PQC candidates is based on the use of quasi-cyclic moderate density parity-check (QC-MDPC) codes [Mis+13], instead of Goppa codes. These code-based schemes improves McEliece in several aspects, one being the simplified description of a code-based PKC without a permutation matrix; another one being the reduced size of the public key. Another key factor in favor of MDPC codes is that they allow for relatively efficient decoding for a legitimate user while still ensuring a required level of security for the corresponding public key. Based on this new idea of using iterative decoding of MDPC codes, a number of different schemes have since then been published. In particular, the NIST PQC standardization process' third round submission schemes BIKE [Ara+17] and HQC [Mel+18] are relevant this paper.

On the flip side of the coin, much effort is required in both lowering the DFR[1] of the iterative decoding process and in increasing the number of correctable errors. Failing to do so affects the legitimate user because it raises the requirement of the block-size (and therefore communication bandwidth) but more importantly, a poor DFR performance enables reaction attacks [GJS16; GJW18; NJW19] based on the observation of uncorrectable error patterns. One decoder is the original iterative Bit-Flipping (BF) decoder [Gal68] and it, when applied to QC-MDPC codes, suffers from relatively poor DFR performance due to it originally being developed for the low-density parity-check (LDPC) codes that are intensively employed in communication systems. The inefficiency of the BF decoder stems from the higher probability of MDPC codes to include short cycles in the corresponding Tanner graph [Mis+13].

A number of different attempts to improve BF decoding of QC-MDPC codes have been developed in the last couple of years. In particular, the Black-Grey Flip (BGF) decoder [DGK19] was selected by BIKE due to its quick convergence (i.e., good DFR despite a small number of iterations).

Similarities between LDPC codes and MDPC codes allow for a transfer of ideas which lie behind some improved BF algorithms to the cryptography area. However, specific features of MDPC codes require significant modifications of known iterative decoding techniques. Examples of improved BF algorithms of this type can be found in [BKS19] and [BL18], where BF decoding followed by post-processing was studied.

Our main contribution in this paper is a new decoder design following an analysis of belief propagation (BP) decoding. Due to the increased computational complexity of our decoder, we additionally present a hybrid of our design and that of the BGF decoder. Both new decoders are evaluated with simulations to confirm their respective DFRs.

[1] Decoding Failure Rate, also known as the Frame Error Rate (FER) in the field of coding theory.

2 QC-MDPC based McEliece cryptosystem

Key Generation, Encryption and Decryption can be performed in a number of different ways (e.g., see BIKE specification [Ara+20]), but for the sake of simplicity and generality we present a summary of the original proposal by Misoczki, Tillich and Sendrier [Mis+13]. We assume that differences between different variants of QC-MDPC-based schemes do not affect the relative DFR and/or computational efficiency of decoding algorithms, although the performance in absolute terms is expected to depend on the instantiated scheme in question.

We consider the (n, r, w)-QC-MDPC code, where $n = n_0 r$. It is defined through a parity check matrix

$$H = \begin{pmatrix} H_0 & H_1 & \cdots & H_{n_0-1} \end{pmatrix}, \tag{1}$$

where the $r \times r$ submatrices H_i, $i = 0, 1, \ldots, n_0 - 1$ are circulant matrices with row weight w_i. Notice that for MDPC codes $\sum_{i=0}^{n_0-1} w_i$ is chosen to be of order $\sqrt{n \log n}$. In the sequel, for simplicity, we analyze only the case of $n_0 = 2$ and equal weights $w_i = w$, for $i = \{0, 1\}$.

1. *Key Generation:* Generate two random sequences $\boldsymbol{h}_0, \boldsymbol{h}_1$ of length $r = n/2$ and Hamming weight w. Form circulant matrices H_0, H_1 as cyclic shifts of $\boldsymbol{h}_0, \boldsymbol{h}_1$, respectively.

   Construct a generator matrix G as

   $$G = \begin{pmatrix} I \mid Q \end{pmatrix}, \text{where } Q = \left(H_1^{-1} H_0\right)^{\mathrm{T}}, \tag{2}$$

   where T denotes matrix transposition and I is the identity matrix. The public key is G (compactly represented by the first row of Q) and the private key is H (compactly represented by the sequence $\boldsymbol{h}_0, \boldsymbol{h}_1$).

2. *Encryption:* Generate a random sequence $\boldsymbol{e}$ of length n and Hamming weight t. The ciphertext $\boldsymbol{x}$ for the plaintext $\boldsymbol{m}$ is $\boldsymbol{x} = \boldsymbol{u} + \boldsymbol{e}$, where $\boldsymbol{u} = (\boldsymbol{m}, \boldsymbol{m}Q)$.

3. *Decryption:* Perform iterative decoding of $\boldsymbol{x}$ by using the known matrix H from Eq. (1). Obtain the plaintext $\boldsymbol{m}$ as the first k bits of the decoded sequence.

3 Decoding of MDPC Codes

In this section, we briefly recall the iterative decoding algorithms applied to decoding LDPC/MDPC codes. There are many implementations of the BF decoding [Gal63], which differ both in complexity and error correcting performance. For the detailed overview of MDPC-code-based versions see [MOG15; DGK19]. All of these versions are used in the same manner as for LDPC codes, determined by sparse parity-check matrices. Our motivation is to modify the BF algorithm in order to take into account that parity-check matrices of MDPC codes are much denser than those of LDPC codes.

3.1 Belief propagation decoding

We start with describing the soft-decision BP algorithm, which shows close to optimal performance when used with LDPC codes. In decryption, we interpret vectors $\boldsymbol{u}$ and $\boldsymbol{x} = (x_1, \ldots, x_n) = \boldsymbol{u} + \boldsymbol{e}$ as the input and output of the binary symmetric channel (BSC) with crossover probability $\epsilon = t/n$. In order to apply BP decoding, we transform the hard-decision BSC output to a soft-decision sequence of log-likelihoods ratios (LLRs) $\boldsymbol{y} = (y_1, \ldots, y_n)$, where

$$y_i = \mu(1 - 2x_i), \ \mu = \log \frac{1-\epsilon}{\epsilon}. \tag{3}$$

BP decoding (alternatively called message passing algorithm) has many different implementations. Two of them, namely, sum-product (SP) algorithm [Gal63] and its approximation, called min-sum (MS) algorithm [FMI99] are well known in coding theory.

Let $V = \{1, \ldots, n\}$ and $C = \{1, \ldots, r\}$ be the sets of variable and check nodes (code symbols and parity-checks) of an MDPC code's Tanner graph and denote by $Z = \{z_{cv}\}$ messages exchanged between nodes $c \in C$, $v \in V$. Notice that number of elements in Z is determined by the number of non-zero elements in the parity-check matrix H.

SP and MS decoding

1. Initialization: let $z_{cv} = y_v$, $v \in V$, $c \in C$.
2. At each iteration of BP decoding
   - For each $c \in C$ and all v connected to c compute either

$$z_{cv} = \prod_{v' \neq v} \text{sign}(z_{cv'}) \phi \left(\sum_{v' \neq v} \phi(|z_{cv'}|) \right), \tag{4}$$

where $\phi(\lambda) = \ln \frac{e^\lambda + 1}{e^\lambda - 1}$, $\lambda > 0$. or

$$z_{cv} = \prod_{v' \neq v} \text{sign}(z_{cv'}) \alpha \left(\min_{v' \neq v} |z_{v'}| - \beta \right), \tag{5}$$

for SP and MS versions, respectively. The constants $\alpha > 0$ and $\beta \geq 0$ represent a normalization factor and an offset, respectively.

- For each $v \in V$ and all c connected to v update

$$z_{cv} = y_v + \sum_{c\prime \neq c} z_{c\prime v}. \tag{6}$$

3. Compute hard decisions as

$$u_v = \text{sign}\left(y_v + \sum_{c} z_{cv}\right). \tag{7}$$

Early termination of the decoding procedure can be done if at each iteration the sequence of hard decisions is computed according to Eq. (7) and the corresponding syndrome is evaluated. All-zero syndrome vector indicates that a valid codeword is recovered. The complexity of the SP version of BP decoding is determined by the computational complexity of Eq. (4).

Bit-Flipping (BF) decoding

BF decoding is a simple hard-decision version of BP decoding algorithm. The two originally suggested in [Gal63]. BF algorithms differ in the way the decoding threshold is chosen.

1. Initialization: compute syndrome $\boldsymbol{s} = \boldsymbol{x}H^{\mathrm{T}}$.
2. At each iteration
   - For each $c \in C$ and all v connected to c set $z_{cv} = s_c$, where s_c is the c-th component of $\boldsymbol{s}$.
   - For each $v \in V$ and all c connected to v compute a number of unsatisfied checks (NUC): $\sigma_v = \sum_c z_{cv}$.
   - Flip symbols of $\boldsymbol{x}$ at positions v such that $\sigma_v > \max_v \sigma_v - \delta$.
3. Return $\boldsymbol{x}$.

Early termination can be performed if none of positions were flipped at some iteration. The parameter $\delta > 0$ determines a tradeoff between DFR and complexity.

3.2 Black-Grey-Flip Decoder (BGF)

Building on the basic BF (Bit-Flipping) decoder we first briefly describe the "Black-Grey" (BG) decoder[2] described in [DGK20b]. On top of the BF algorithm this

[2] "Black-Grey-Flip" (BGF) is an optimized variant of the BG decoder first found in the BIKE pre-Round-1 submission CAKE by Sendrier and Misoczki.

decoder utilizes two additional lists (called black and grey, respectively) to keep track of bit flips which are considered "uncertain". The bits specified in these lists are reevaluated against conditions. Bits that meet these conditions are flipped again, thereby undoing the first flip. Each iteration of the BG implementation is divided into three steps.

Step I Perform a single round of BF decoding, (i.e. flipping bits according to their NUC value σ_v and a threshold τ) producing two lists:

- A black list containing all flips performed.
- A grey list containing all flips with a NUC value close to (but not above) the threshold.

Recompute the syndrome.

Step II Another round of BF decoding but this time only consider bits in the black list. Recompute the syndrome.

Step III Another round of BF decoding but this time only consider bits in the grey list. Recompute the syndrome.

BGF introduces an optimization due to recognizing (see [DGK20b]) that it is only during the first iteration there are uncertain bits to reevaluate. Steps II and III are therefore only performed for the first iteration. Leaving only Step I makes BGF equivalent to BF decoding during the remaining number of iterations. One can view the relative time-complexity of the BGF decoder as $1 \times \text{BG}_{\text{rounds}} + 4 \times \text{BF}_{\text{rounds}} \approx 7 \times \text{BF}_{\text{rounds}}$ according to [DGK20b].

4 Analysis of BF decoding of MDPC codes

In BP decoding (SP, MS or BF) we associate with nonzero elements of parity-check matrix H an array of random variables z_{cv}. At decoding iterations, z_{cv} values are updated depending on syndrome values. In case of BF decoding the Hamming weight of parity checks determines row-processing errors to next iterations. In what follows, we try to construct an approximate model of error propagation in BP decoding.

For a code of rate R defined by a parity-check matrix H with column weight w, a single error influences w rows with $d = w/(1-R) = wn_0$ nonzero symbols in each row. In total, $N_1 = wd$ variables z_{cv} associated with nonzero elements of H in w parity checks are affected by a single error. Among them only w elements belong to the column corresponding to the "true" erroneous variable node. The other $(wd - w)$ non-zeros can be interpreted as an additional binary noise caused by correlation between parity checks.

If the number of introduced errors $t \gg 1$, then the nonzero symbols of the binary noise are assumed to be independent and identically distributed. Assume that a single variable z_{cv} is equal to 1 due to a single channel error with probability

$$p = \frac{\text{\#of affected positions}}{\text{total \# of nonzero elements}} = \frac{wd - w}{rd} = \frac{w - 1 + R}{r} \ .$$

If t errors occur then z_{cv} will be equal to 1 if the number of its flips will be odd and equal to 0, otherwise. The probability of "false" ones caused by odd number of flips is equal to (see [Gal63])

$$\varepsilon = \frac{1 - (1-2p)^t}{2} = \frac{1 - \left(\frac{2(w-1+R)}{r}\right)^t}{2}. \tag{8}$$

Below we interpret ε in Eq. (8) as the probability of errors in computing NUCs induced by the binary noise caused by high correlation of parity checks.

Example 1. *Let* $r = 4801$*,* $w = 45$*,* $t = 100$*. It is easy to check that* $\varepsilon = 0.423$ *and that the expected value of NUC is equal to* $\epsilon w = 18.9$ *for error-free positions and is equal to* $(1-\epsilon)w = 26.1$ *for positions in error. These estimates were confirmed by long simulations.* □

What follows from this example is that the information about errors obtained from NUC values σ_v is "extremely noisy". We can expect $\sigma_v < w/2$ for many error positions and $\sigma_v > w/2$ for many error-free positions In other words, syndrome values used in BF decoding for making decisions about bits to be flipped are typically unreliable. These considerations lead us to the conclusion that syndrome values used in BF decoding should be classified and weighted according to their 'reliabilities'. Similar ideas are behind one improvement of BF decoding in [KK10].

5 The new versions of BF decoder

For a given syndrome vector $\boldsymbol{s} = (s_0, \ldots, s_{r-1})$, particular values $s_c = 0$ or 1 mean that the number of erroneous (to be flipped) positions θ_c among positions involved into the c-th check is even or odd, respectively. Our goal is to find a statistical criterion for distinguishing between cases $\theta_c = 0$ or $\theta_c \in \{2, 4, \cdots\}$ if $s_c = 0$ and between cases $\theta_c = 1$ or $\theta_c \in \{3, 5, \cdots\}$ if $s_c = 1$. In other words, we aim at estimating "reliability" of the syndrome components s_c.

Positions v with the number of NUCs $\sigma_v > w/2$ are considered as candidates for flipping. For a check c, $c = 0, \ldots, r-1$ let $\boldsymbol{\sigma}^{[c]} = (\sigma_0^{[c]}, \ldots, \sigma_{d-1}^{[c]})$ be a sequence of $\sigma_i = \sigma_{v_i}$, where v_i, $i = 0, \ldots, d-1$ are nonzero symbols of the check c. We are going to measure reliability of s_c as a function of the NUC sequence $\boldsymbol{\sigma}^{[c]}$.

Let $\chi(x)$ be an indicator function which is equal to 1 if x is true and 0, otherwise. The number of votes for flipping in the c-th parity check can be expressed as

$$\theta_c = \sum_{i=0}^{d-1} \chi(\sigma_i^{[c]} > w/2). \tag{9}$$

To obtain a quantitative measure of reliability we evaluated, empirically for the MDPC code from Example 1, the following logarithmic likelihood values

$$\begin{aligned} rClA_0(\theta_c) &= \log \frac{\Pr(e_c = 0|s_c = 0, \theta_c)}{\Pr(e_c > 0|s_c = 0, \theta_c)}, \qquad (10)\\ A_1(\theta_c) &= \log \frac{\Pr(e_c = 1|s_c = 1, \theta_c)}{\Pr(e_c \neq 1|s_c = 1, \theta_c)}, \qquad (11)\end{aligned}$$

where e_c is the number of errors (out of t introduced errors) in the symbols of the c-th check. Functions $A_s(\theta_c)$, $s = 0, 1$ are identical for all checks $c = 0, \ldots, r-1$ due to the code regularity. This allows to use $A_s(\theta)$ as a reliability measure for the syndrome value s given that the number of votes for flipping in a check is equal to θ. Functions $A_s(\theta)$ empirically evaluated for the code of Example 1 are presented in Fig. 1. We can see that to the syndrome components s corresponding to the check with a small number of votes θ should be assigned 3–4 times larger weights than to the components corresponding to checks with large θ.

It follows that after computing syndrome and NUCs, one can compute weighting factors for syndrome component either by A_0 or by A_1 depending on the value of the syndrome component. For all variable nodes, the weighted NUC values can be obtained by adding the reliability info with NUC weighted by 1/2 (chosen empirically). The rest of the algorithm works as in the original BF algorithm with the threshold equal to the maximal shifted and weighted NUC value. A variable node with the maximal value of combined reliability and NUC is flipped.

The algorithm shown in Fig. 2 is an approximated version of the theoretical WBF algorithm described above. Instead of using fixed arrays of weighting coefficients A_0, A_1, the algorithm searches for minimum numbers of votes separately for satisfied and unsatisfied checks (lines 6, 7). The contribution of syndrome components with the smallest number of votes is amplified (lines 10 and 12) by multiplication with the fixed coefficient 3.

Simulations show that performance of the theoretical and simplified weighted algorithms are hardly distinguishable.

In Fig. 2, only one position is flipped per iteration. Similarly to the BF decoding, a certain threshold can be selected in such a way that at step 16 in Fig. 2 more than one flip can be allowed. This modification will make the convergency faster at the cost of slightly increasing DFR. The simulated WBF decoder in Section 6.2 use this optimization.

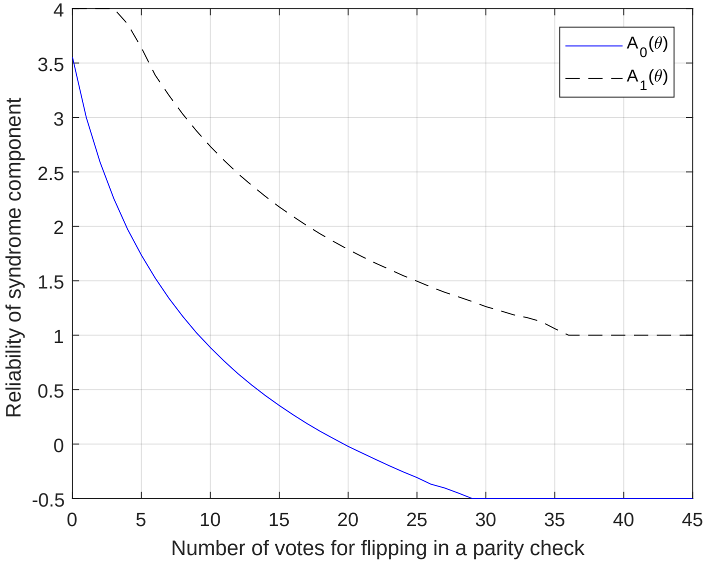


Figure 1: Logarithmic likelihood ratios defined as in Section 5 for syndrome components as functions of the number of votes for flipping (NUCs above $w/2$)

```
function [y,it]= WBF(y,V,C,w,maxit)
s=mod(sum(y(V),2),2)'; % syndrome
for it=1:maxit
  nuc=sum(s(C));         % NUC
  SM=sum(nuc(V)>w/2,2)'; % votes
  minSM1=min(SM(s==1));
  minSM0=min(SM(s==0));
  REL=2*s-1;
  mask=(s==0) & (SM<=minSM0+3);
  REL(mask==1)=REL(mask==1)*3;
  mask=(s==1) & (SM<=minSM1+3);
  REL(mask==1)=REL(mask==1)*3;
  metric=sum(REL(C));
  [mm,~]=max(metric);
  f=find(metric==mm);
  [~,J]=min(sum(SM(C(:,f))));
  s(C(:,f(J)))=~s(C(:,f(J)));
  y(f(J))=~y(f(J)); % flip
  if sum(s)==0, return; end
end
```

Figure 2: Matlab program for simplified WBF decoding.

5.1 Complexity and constant-time considerations

A single round of normal BF decoding comprises of the following parts:

- (Re)compute the syndrome
- Check the syndrome Hamming weight.
- Calculate NUCs.
- Check NUCs and flip based on the threshold.

It has been shown to be possible to implement these operations efficiently and in constant time [DG19]. For WBF decoding we must *additionally* consider the following operations:

- Calculate the number of votes to flip, for each syndrome position.
- Find the minimum number of votes. [3]
- Determine the reliability of each syndrome value (based on number of votes).
- Determine a new "metric" for each variable node (based on the weighted syndrome values).
- Find and flip the variables with the highest metric.

It is possible to implement these operations in a constant time manner, since none of them requires branching or indexing based on secret data. However, to implement them *efficiently* is left as out of scope for this paper.

5.2 A hybrid decoder

Accounting for the decrease in interference noise due to a reduction in the number of error positions, we know that making the right bit-flipping decisions is more important early in the decoding process. Due to this, one can argue for an optimization of using an adaptable decoding algorithm, which progressively uses less and less expensive computations (this is exactly the optimization introduced for the BG decoder to make the BGF decoder) [DGK20b].

To realize such an algorithm while keeping the implementation complexity low and constant time we look into a hybrid solution where we pair two different decoding algorithms based on DFR and execution speed criteria.

We consider the pairing of our WBF algorithm together with Black Grey Flip (BGF) decoder, which has a very quick convergence. We limit WBF to run only

[3] It might be possible to empirically determine a static threshold to use instead of dynamically calculate the minimum.

2 iterations, after which we run the (BGF) decoder for 3 iterations. As previously explained, this means that the BGF decoder will run a single round of BG decoding followed by 2 rounds of normal BF decoding. The complexity of the combined algorithm can informally be thought of as $2 \times \text{WBF}_{\text{rounds}} + 1 \times \text{BG}_{\text{rounds}} + 2 \times \text{BF}_{\text{rounds}} \approx 2 \times \text{WBF}_{\text{rounds}} + 5 \times \text{BF}_{\text{rounds}}$. Appreciating the fact that one round of WBF requires more computations than one round of BF we recognize that even this hybrid will be slower than that of the BGF decoder. Our hybrid decoder can still remain a competitive alternative if properly implemented (e.g., by using techniques from [DG19]).

6 Simulation

In this section, we present a comparative analysis between our proposed decoders and the BGF decoder.

6.1 Method

We apply a similar methodology as the one given in [SV20; SV19; DGK20b; DGK20a]. By using the same parameters as those used by BIKE (security level 1) [Ara+20], we facilitate easier comparisons of different DFR rates. One must note that we use a different way of encoding messages, which might affect the DFR rates in absolute terms. However, the relative differences between decoders are assumed to be unaffected, as previously stated.

6.2 Results

In this section, we present the simulation results of the decoding algorithms studied in this paper. The data is presented in Fig. 3. Using the same methodology as in [SV20; SV19; DGK20b; DGK20a], we attempt in Table 1 to estimate a value of r that achieves a DFR $= 2^{-128}$. The estimation was made using a simple linear extrapolation from the last two data points of each graph. This is the most straightforward and conservative choice [SV20].

Table 1: A summary of the different decoding algorithms. See text for extrapolation method.

Algorithm	Max # Iterations	Average # Iterations	Extrapolated r (DFR $= 2^{-128}$)
WBF	92	34	13379
Hybrid	5	5	13018
BGF	5	5	13660

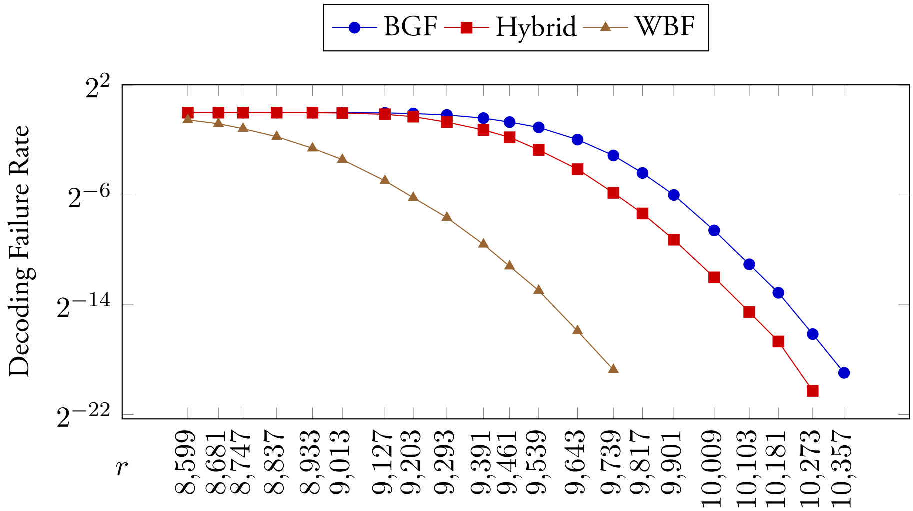


Figure 3: Graph of the DFR as a function of r, for different decoding algorithms. Minimum 1,000 decryption failures for each data point with each key used for maximum 10,000 ciphertexts.

In Table 1 the extrapolation does not yield quite the expected values. For instance, the Hybrid decoder performs better than WBF, although the opposite is clearly visible in Fig. 3. We assume this to be a limitation of the experiment; even lower DFRs are needed if we want to make certain that we are indeed extrapolating only the linear part of the curves. The WBF and hybrid algorithms are also fine-tuned for lower r-values with a high DFR, due to practical reasons. It is possible that the WBF algorithm can be further tuned for r-values corresponding to a low DFR.

7 Conclusion

In this paper, we presented a new Weighted Bitflipping iterative decoder (WBF). We showed that it significantly lowers the DFR for all simulated parameters compared to other decoders. Specifically, we compared it to the Black Grey Flip decoder as found in the NIST PQC 3rd round submission BIKE [Ara+20].

Due to the increased computational complexity of the WBF decoder, we proposed a hybrid decoder which combines the quick convergence of the BGF decoder with the better DFR of the WBF decoder. This results in a much smaller computational complexity by lowering the required number of iterations.

Acknowledgments

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, by the grant PRG49 from the Estonian Research Council and by the ERDF via CoE project EXCITE. The computations were enabled by resources provided by the Swedish National Infrastructure for Computing (SNIC) at LUNARC, partially funded by the Swedish Research Council through grant agreement no. 2018–05973.

References

[Ara+17] N. Aragon, P. Barreto, et al. “BIKE: bit flipping key encapsulation”. In: (2017).

[Ara+20] N. Aragon, P. S. L M Barreto, et al. *BIKE: Bit Flipping Key Encapsulation Submission for Round 3 Consideration*. Tech. rep. 2020.

[BKS19] I. E. Bocharova, B. D. Kudryashov, and V. Skachek. “AVN-based Elimination of Short Cycles in Tanner Graphs of QC LDPC Codes”. In: *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE. 2019, pp. 56–60.

[BL18] H. Bartz and G. Liva. “On Decoding Schemes for the MDPC-McEliece Cryptosystem”. In: *arXiv preprint arXiv:1801.05659* (2018).

[Bos+17] J. Bos et al. *CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM*. Cryptology ePrint Archive, Report 2017/634. `https://eprint.iacr.org/2017/634`. 2017.

[DG19] N. Drucker and S. Gueron. “A toolbox for software optimization of QC-MDPC code-based cryptosystems”. In: *J. Cryptogr. Eng.* 9.4 (Nov. 2019), pp. 341–357.

[DGK19] N. Drucker, S. Gueron, and D. Kostic. *On constant-time QC-MDPC decoding with negligible failure rate*. Tech. rep. Cryptology ePrint Archive, Report 2019/1289, 2019.

[DGK20a] N. Drucker, S. Gueron, and D. Kostic. “On Constant-Time QC-MDPC Decoders with Negligible Failure Rate”. In: *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*. Vol. 12087 LNCS. 2020, pp. 50–79.

[DGK20b] N. Drucker, S. Gueron, and D. Kostic. *QC-MDPC decoders with several shades of gray*. Tech. rep. Report 2019/1423. 2020, pp. 1–16.

[FMI99] M. P. Fossorier, M. Mihaljevic, and H. Imai. "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation". In: *IEEE Trans. on Commun.* 47.5 (1999), pp. 673–680.

[Gal63] R. G. Gallager. *Low-density parity-check codes*. M.I.T. Press: Cambridge, MA, 1963.

[Gal68] R. G. Gallager. *Information theory and reliable communication*. Wiley, 1968.

[GJS16] Q. Guo, T. Johansson, and P. Stankovski. "A key recovery attack on MDPC with CCA security using decoding errors". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2016, pp. 789–815.

[GJW18] Q. Guo, T. Johansson, and P. S. Wagner. "A Key Recovery Reaction Attack on QC-MDPC". In: *IEEE Trans. on Inf. Theory* (2018).

[KK10] H. Kamabe and S. Kobota. "Simple improvements of bit-flipping decoding". In: *2010 The 12th International Conference on Advanced Communication Technology (ICACT)*. Vol. 1. IEEE. 2010, pp. 113–118.

[McE78] R. J. McEliece. "A public-key cryptosystem based on algebraic". In: *Coding Thv* 4244 (1978), pp. 114–116.

[Mel+18] C. A. Melchor et al. "Hamming quasi-cyclic (HQC)". In: *NIST PQC Round* 2 (2018), pp. 4–13.

[Mis+13] R. Misoczki, J.-P. Tillich, N. Sendrier, and P. S. Barreto. "MDPC-McEliece: New McEliece Variants from Moderate Density Parity-Check Codes". In: *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*. IEEE. 2013, pp. 2069–2073.

[MOG15] I. v. Maurich, T. Oder, and T. Güneysu. "Implementing QC-MDPC McEliece Encryption". In: *ACM Transactions on Embedded Computing Systems (TECS)* 14.3 (2015), p. 44.

[NJW19] A. Nilsson, T. Johansson, and P. Wagner Stankovski. "Error Amplification in Code-based Cryptography". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* (2019), pp. 238–258.

[SV19] N. Sendrier and V. Vasseur. "On the Decoding Failure Rate of QC-MDPC Bit-Flipping Decoders". In: 11505 LNCS (2019), pp. 404–416.

[SV20] N. Sendrier and V. Vasseur. "About low DFR for QC-MDPC decoding". In: *International Conference on Post-Quantum Cryptography*. Springer. 2020, pp. 20–34.

Don’t Reject This: Key-Recovery Timing Attacks Due to Rejection-Sampling in HQC and BIKE

Abstract

Well before large-scale quantum computers will be available, traditional cryptosystems must be transitioned to post-quantum (PQ) secure schemes. The NIST PQC competition aims to standardize suitable cryptographic schemes. Candidates are evaluated not only on their formal security strengths, but are also judged based on the security with regard to resistance against side-channel attacks. Although round 3 candidates have already been intensively vetted with regard to such attacks, one important attack vector has hitherto been missed: PQ schemes often rely on *rejection sampling* techniques to obtain pseudorandomness from a specific distribution. In this paper, we reveal that rejection sampling routines that are seeded with secret-dependent information and leak timing information result in practical key recovery attacks in the code-based key encapsulation mechanisms HQC and BIKE.

Both HQC and BIKE have been selected as alternate candidates in the third round of the NIST competition, which puts them on track for getting standardized separately to the finalists. They have already been specifically hardened with constant-time decoders to avoid side-channel attacks. However, in this paper, we show novel timing vulnerabilities in both schemes: (1) Our secret key recovery attack on HQC requires only approx. 866,000 idealized decapsulation timing oracle queries in the 128-bit security setting. It is structurally different from previously

Q. Guo, C. Hlauschek, T. Johansson, N. Lahr, A. Nilsson, and R. L. Schröder. “Don’t Reject This: Key-Recovery Timing Attacks Due to Rejection-Sampling in HQC and BIKE”. in: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2022.3 (June 2022), pp. 223–263 identified attacks on the scheme: Previously, exploitable side-channel leakages have been identified in the BCH decoder of a previously submitted HQC version, in the ciphertext check as well as in the pseudorandom function of the Fujisaki-Okamoto transformation. In contrast, our attack uses the fact that the rejection sampling routine invoked during the deterministic re-encryption of the decapsulation leaks secret-dependent timing information, which can be efficiently exploited to recover the secret key when HQC is instantiated with the (now constant-time) BCH decoder, as well as with the RMRS decoder of the current submission. (2) From the timing information of the constant weight word sampler in the BIKE decapsulation, we demonstrate how to distinguish whether the decoding step is successful or not, and how this distinguisher is then used in the framework of the GJS attack to derive the distance spectrum of the secret key, using 5.8×10^7 idealized timing oracle queries. We provide details and analyses of the fully implemented attacks, as well as a discussion on possible countermeasures and their limits.

1 Introduction

The progress in the research field of quantum computing weakens the previously estimated security guarantees of most currently deployed cryptographic primitives. In 2017, Michele Mosca [Mos17] estimated that the chance of having a large-scale quantum computer that breaks RSA-2048 to be $1/6$ within a decade and $1/2$ within 15 years; or even faster (6-12 years) by having massive investment, following Simon Benjamin [Ben17]. While such estimates and predictions are contested [Dya18; Kal20], it is important that the transition to post-quantum secure cryptographic algorithms happens well before an actual large-scale quantum computer is being built, as sensitive data might be stored for cryptanalysis at a later time, for example by surveillance infrastructure such the NSA's 3-12 exabyte data center in Utah [Hog15].

The security strengths of the new cryptographic primitives need to be evaluated with regard to possible attacks from classical as well as from quantum adversaries. But not only the algorithmic design need to withstand possible (theoretical) attacks, deployed schemes need to have secure implementations that withstand practical implementations attacks [HPA21], such as side-channel [Koc96; KJJ99; Rav+04] and fault attacks [BDL97; BDL01]. Not every cryptographic design has a straightforward elegant implementation that can be easily secured against all relevant implementation attacks. Daniel Bernstein and Tanja Lange repeatedly (e.g., in their analysis of the NIST ECC standards [BL16]) emphasize that a good cryptographic design requires simplicity of a secure implementation, and recommend that standardization bodies such as the National Institute of Standards and Technology (NIST) should require simplicity for secure implementations.

Timing attacks, first described by Kocher [Koc96], are arguably one of the most dangerous implementation attacks (right after more trivial, but still hard to spot, leakages such as the Heartbleed vulnerability [Dur+14]): an adversary just needs a communication channel to the target device and a precise timing measurement. It is often possible to mount an attack even remotely over the network [BB05; BT11; Kau+16; Mog+20; Mer+21], without physical access. Crosby et al. [CWR09] explore the limits of remote timing attacks. Often, timing leaks that have been mitigated against remote exploitation, such as the Lucky Thirteen attack [AP13] on TLS, can still be exploited in a Cloud/Cross-VM setup [Ape+15]. These attacks exploit the timing variations which depend on the secret key material. When the timing variations include enough information the recovery of the secret key becomes possible.

In December 2016, the NIST announced a competition [SN16] which aims to standardize schemes for post-quantum cryptography (PQC) and encouraged the authors to submit a reference implementation that addresses side-channel attacks in addition to the specification. NIST specifically motivates research to counter advanced side-channel attacks in the current, third round of the competition [Moo+20]. The two schemes Hamming Quasi-Cylic (HQC) [Agu+21] and the Bit Flipping Key Encapsulation (BIKE) [Ara+21] are promising code-based key encapsulation schemes and alternate candidates in the third round of the competition. As alternate candidates, they might be standardized by NIST in addition to the competition finalists in a fourth round. In its latest PQC standardization status report [Moo+20], NIST lauds HQC for its constant-time improvements, while voicing serious concerns over BIKE's side-channel protections and Indistinguishability under Chosen Ciphertext Attack (IND-CCA) security.

BIKE can be described as the McEliece scheme instantiated with Quasi-Cyclic Moderate Density Parity-Check (QC-MDPC) codes [Mis+13], using the equivalent Niederreiter scheme. The specification of BIKE is secure under the Indistinguishability under Chosen Plaintext Attack (IND-CPA) notion, where the security is related to some hard decoding problems in the Hamming metric. With an additional assumption on the probability of decryption errors that may occur in the BIKE decoding step, the scheme is shown to be IND-CCA secure, using the implicit-rejection variant of the Fujisaki-Okamoto (FO) transformation proposed by Hofheinz, Hövelmanns, and Kiltz (HHK) [HHK17]. Similarly, the Public Key Encryption (PKE) variant of HQC is secure under the IND-CPA notion. The Key Encapsulation Mechanism (KEM) variant of HQC utilizes another variant of the HHK FO transformation, converting the PKE variant to be secure with regard to IND-CCA. Many post-quantum secure schemes, e.g., code- and lattice-based, use the HHK transformation because it is resistant to the decryption errors that can occur in the decryption procedure of many non-traditional schemes. The attacks on BIKE as well as on HQC demonstrated in this paper are both possible due to the specific applications of the FO transformation to the respective underlying IND-CPA schemes. It is interesting to observe that the process of transforming an encryption scheme from a less secure to a more secure version introduces more complexity and hard-to-spot vectors for additional implementation vulnerabilities.

However, our attacks require a static key setting and an active chosen-ciphertext attacker, a scenario where one would not employ just an IND-CPA secure scheme.

Related work. Our attacks, though structurally different from previous ones, build on a history of related side-channel attacks and cryptanalysis, leading to incrementally more secure and improved versions of the schemes. Recently, Wafo-Tapa et al. [Waf+19] and Paiva et al. [PT19] present timing attacks on the non-constant time implementation of the Bose-Chaudhuri-Hocquenghem (BCH)-decoder of HQC. Both approaches exploit the dependence between the running time of the decoding procedure and the number of decoded errors. Paiva et al. require $400 \cdot 10^6$ decryption runs for the 128-bit security parameters. Wafo-Tapa et al. reach a key recovery after just 5441 calls with 93% success rate for the same security level. They proposed a constant-time BCH decoding to fix this issue.

Guo et al. [GJN20] show that the FO transformation of various proposed schemes is vulnerable to a timing attack by exploiting the comparison step in the decapsulation function, which is usually non-constant time (for example, when implemented via the `memcmp` function of the standard C library). The authors apply this timing attack to the lattice-based scheme FrodoKEM [Nae+20]. The attack requires 2^{30} decapsulation calls. They state that their attack is applicable to other proposed PQC schemes, among others, to HQC. They show the applicability to LAC [Lu+19] in the appendix but do not explicitly show the effectiveness to HQC. The countermeasure to avoid the leakage is to use another constant-time comparison, e.g., as provided by OpenSSL[1]. However, in the same paper, Guo et al. describe how, more generally, any timing variance in the FO transformation that allows to distinguish between modifications that are below or above the error correction capability of the underlying primitive can in principle be used to mount key recovery attacks on IND-CCA secure KEM schemes. The research community seems to be acutely aware of the need to implement FO-based decapsulation methods in a constant time manner, as the source code of both HQC and BIKE, as well as recent discussions on the NIST PQC Forum[2] indicate.

An important attack for schemes based on QC-MDPC codes such as BIKE is the GJS attack [GJS16]. The attack uses an identified dependence between error patterns in decoding failures and the secret key. This attack assumes that the scheme is used in a static key setting requiring IND-CCA security. The *Error Amplification* attack [NJW18] builds on the GJS attack [GJS16], but improves it by using only a single initial error vector that results in a decoding failure and then modifies this in order to efficiently generate many more error vectors causing a decoding failure. These attacks are avoided in the BIKE scheme by selecting parameters such that the probability of a decoding failure for properly generated ciphertexts is very small.

[1] `https://www.openssl.org/docs/man1.1.1/man3/CRYPTO_memcmp.html`

[2] `https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/iVbJkCytoog`

Most recently, Ueno et al. [Uen+22] explore a generic side-channel attack of the FO transformation commonly used in many PQC schemes: by exploiting side-channel leakage during non-protected Pseudorandom Function (PRF) execution in the re-encryption of the KEM decapsulation, they demonstrate that Kyber, Saber, FrodoKEM, NTRU, NTRU Prime, BIKE, SIKE, as well as HQC are vulnerable. The current reference implementation of HQC uses non-protected SHAKE as the relevant PRF.

Motivation. To the best of our knowledge, none of the previous attacks on HQC or BIKE, nor attack mitigations for those schemes, considered the non-constant time rejection sampling routine. Rejection sampling is one method to generate pseudorandom values from a specific distribution. The first side-channel attack targeting rejection sampling has been demonstrated at the CHES 2016 against the lattice-based signature BLISS [Bru+16], exploiting cache-access pattern from the Gaussian sampler. In contrast to the Gaussian samplers common in lattice-based schemes, HQC and BIKE use rejection sampling to generate random vectors with a specific Hamming weight. As we show in this paper, it turns out that the branching and thus the run-time of the rejection sampling routine in HQC and BIKE is indirectly in dependence relationship with the key.

This despite the fact that the current HQC specification states that the optimized reference implementation (using the vectorized Single Instruction Multiple Data (SIMD) instructions on an x86 machine) is now constant-time, and the source code is well analyzed concerning the leakage of any sensitive information. More specifically, the authors of HQC claim "to have thoroughly analyzed the code to check that only unused randomness (i.e. rejected based on public criteria) or otherwise nonsensitive data may be leaked." [Agu+21]. However, the specification reveals a subtle error: the modular design of the HQC KEM uses the FO transformation to transform an IND-CPA version of HQC into the IND-CCA KEM. This IND-CPA version is specified separately as a non-deterministic encryption scheme, where the Encrypt algorithm generates its randomness within the function. The specified KEM version then invokes a slightly different HQC.PKE encryption scheme that fixes the randomness via parameter passing to make the encryption deterministic. This subtle error in the specification might have hidden the fact that the rejection sampling invoked by the re-encryption step in the decapsulation routine has a dependence to the secret key: from reading this erroneous specification, it is easy to miss the fact that the rejection sampling in the Encrypt function is indeed dependent on the secret in the decapsulation. Adding further to the confusion, the plaintext message **m** can be chosen by the attacker in the IND-CPA scheme, and only becomes a secret-depending value in the IND-CCA KEM due the FO transformation.

On the other hand, the BIKE specification [Ara+21] demands (in the current version 4.2a, in Section 3.5 *Practical security considerations for using BIKE*) with regard to side-channel attacks only that the decoder must be implemented in constant time, but does not mention such considerations for the constant-weight hashing function, which is supposed to be implemented via rejection sampling.

While HQC uses non-constant time rejection sampling to generate pseudorandom vectors with a specific Hamming weight in the re-encryption step of the decapsulation due the employed variant of the HHK FO transformation, BIKE uses the implicit-rejection version of it, optimized so that it does not invoke the encryption function during the decapsulation. However, despite dispensing the re-encryption step, BIKE uses rejection sampling in the plaintext verification of the decryption step in the decapsulation routine.

Contributions. In this work, we analyze the current KEM variant of HQC and BIKE and show that they are still vulnerable to timing attacks. More specifically, we present

- hitherto unconsidered timing variations dependent on the secret key in the deterministic re-encryption of the KEM decapsulation of HQC, and in BIKE decapsulation in the plaintext-checking step of the decryption routine, both due to the non-constant time *rejection sampling* routines,
- a novel timing attack on the optimized reference implementation of HQC achieving a full secret key recovery with high probability,
- another novel timing attack on the existing implementations of BIKE achieving either a full secret key recovery or an efficient message recovery, where the key recovery attack uses the framework of the GJS attack to derive the distance spectrum of the secret key, and
- a discussion of possible countermeasures to avoid the identified leakage in the deterministic re-encryption step.

Our attacks are practical as the exploited timing variation is relatively large and the number of required idealized decapsulation timing-oracle calls is small. Idealized means that we assume access to perfect timing information in our attack implementations. Concretely, we assume we know the number of `seedexpander` calls that a decapsulation performed. An evaluation of the attack in different noise environments is not the focus of our work. Depending on the set-up (e.g. clock-speed, local or remote, network latency and jitter) the attack may require a different number of real decapsulations. In certain scenarios state of the art methods such as "timeless" timing attacks [van+20] may be employed to obtain significantly better timing information from remote targets, reducing the number of real decapsulations required.

Timeline and Response. The attack on HQC, first described in the master thesis of one of our co-authors [Sch21] and then later in [HLS21][3], and independently

[3] eprint version 2021/1485/20211115:124514

brought to the attention of NIST by the co-authors from Lund University has been acknowledged by the authors of HQC, who in response started working on more optimized countermeasures [Gab21]. We further identified that a similar side-channel exists in BIKE [Sch21; HLS21]. A concrete attack strategy exploiting the side-channel was discovered concurrently to our work and first published by BIKE co-author Nicolas Sendrier [Sen21]. To the best of our knowledge, we are the first to present a fully implemented version of an attack exploiting the identified side-channel. Also in parallel to the writing of this paper, Nicolas Sendrier [Sen21] presented a different approach than suggested here to mitigate the vulnerability, specific to BIKE: Sendrier first shows that the sampling routine we attack can be slightly biased in BIKE, which allows the use of a novel variant of Fisher-Yates sampling [Knu97, p.145] that is constant-time.

Organization. The remaining of the paper is described as follows. In Section 2, we give the preliminaries as well as a description of the specifications of the two KEM schemes HQC and BIKE, respectively. In Section 3, we explain the identified timing weakness in the functions that use rejection sampling and we describe in detail the full key recovery attack on HQC. In the same section we follow up by describing the details of another type of the attack applied on BIKE. In Section 4, we then present all the evaluation results from actually implementing the two previously described attacks. In Section 5, we discuss possible countermeasures and, finally, Section 6 concludes the paper.

2 Background

In this section, we introduce the background information on the schemes, HQC and BIKE, and the preliminaries that we require to explain our attacks in the following sections.

2.1 Preliminaries

We use a notation that we consider as close as possible to both the notations used by the HQC [Agu+21] as well as the BIKE [Ara+21] specification.

$\mathbb{F}_2$ denotes the binary finite field. Both HQC and BIKE use a cyclic polyomical ring, but with different parameters. So, for an integer $n \in \mathbb{Z}$ (resp., $r \in \mathbb{Z}$ in BIKE), we obtain the ring $\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$, (resp. $\mathcal{R} = \mathbb{F}_2[X]/(X^r - 1)$). Elements in $\mathcal{R}$ will be represented by lower-case bold letters. These elements can be interchangeably considered as row vectors in a vector space over $\mathbb{F}_2$. Respective matrices will be represented by upper case bold letters. For $h \in \mathcal{R}$, let $|\mathbf{h}|$ denote the Hamming weight of a vector or polynomial $\mathbf{h}$.

Table 1: The HQC parameter sets [Agu+21]. The base Reed-Muller code is defined by $[128, 8, 64]$.

	RS-S			Duplicated RM						
Instance	n_1	k	d_{RS}	Mult.	n_2	d_{RM}	$n_1 n_2$	n	ω	$\omega_r = \omega_e$
hqc-128	46	16	31	3	384	192	17,664	17,669	66	75
hqc-192	56	24	33	5	640	320	35,840	35,851	100	114
hqc-256	90	32	49	5	640	320	57,600	57,637	131	149

2.2 Hamming Quasi Cyclic – HQC

HQC is a code-based post-quantum IND-CCA secure KEM. It is an alternate candidate in the third round of the NIST PQC competition [Agu+21]. Our work refers to the recent specification from June 2021. The HQC framework from which HQC stems was introduced by Aguilar et al. [Agu+18]. Its security is reduced to problems related to the hardness of decoding random quasi-cyclic codes in the Hamming metric. The scheme uses a concatenated code $\mathcal{C}$ which combines an internal duplicated Reed-Muller code with the outer Reed-Solomon code. The resulting code has a publicly known generator matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n_1 n_2}$.

The parameters are listed in Table 1 and we explain them in the following. The inner duplicated Reed-Muller code is defined by $[n_2, 8, n_2/2]$ and the outer, shortened Reed-Solomon code (RS-S) by $[n_1, k, n_1 - k + 1]$, with $k \in \{16, 24, 32\}$ depending on the corresponding security level. The concatenated code $\mathcal{C}$ is of length $n_1 n_2$. To avoid algebraic attacks the ambient space of vector elements is of length n which is the first primitive prime greater than $n_1 n_2$. It defines the polynomial quotient ring $\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$.

HQC.PKE. The PKE variant of HQC consists of the Algorithms 1 to 3. The key generation in Algorithm 1 samples the elements $\mathbf{h}$, $\mathbf{x}$, and $\mathbf{y}$ from $\mathcal{R}$ uniformly at random where the Hamming weights of $\mathbf{x}$ and $\mathbf{y}$ are ω. The secret key sk consists of $\mathbf{x}$ and $\mathbf{y}$. The public key pk includes $\mathbf{h}$ and $\mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y}$. The encryption function Algorithm 2 first samples the vectors $\mathbf{e}$ of weight ω_e as well as $\mathbf{r}_1$ and $\mathbf{r}_2$ of weight ω_r. The randomness of the sampling is seeded by the additional input θ. Therewith, the sampling becomes deterministic which is desired for the verification in the later decapsulation function. The ciphertext is a tuple with $\mathbf{u} = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2$ and $\mathbf{v} = \mathbf{m}\mathbf{G} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}$. The term $\mathbf{m}\mathbf{G}$ in ?? 273 corresponds to the encoding procedure of the concatenated code $\mathcal{C}$. It begins with the external Reed-Solomon code which encodes a message $\mathbf{m} \in \mathbb{F}_2^k$ into $\mathbf{m}_1 \in \mathbb{F}_{2^8}^{n_1}$. Then the inner Reed-Muller code encodes each coordinate/byte $m_{1,i}$ into $\bar{\mathbf{m}}_{1,i} \in \mathbb{F}_2^{128}$ using $RM(1,7)$. Finally, $\bar{\mathbf{m}}_{1,i}$ is repeated 3 or 5 times depending on the security parameter to obtain $\tilde{\mathbf{m}}_{1,i} \in \mathbb{F}_2^{n_2}$. Thus, we get $\mathbf{m}\mathbf{G} = \tilde{\mathbf{m}} = (\tilde{\mathbf{m}}_{1,0}, \ldots, \tilde{\mathbf{m}}_{1,n_1-1}) \in \mathbb{F}_2^{n_1 n_2}$. The decryption

Algorithm 1 HQC.KeyGen

```
Input: param
Output: sk, pk
h = Sample($\mathcal{R}$)
x = Sample($\mathcal{R}, \omega$)
y = Sample($\mathcal{R}, \omega$)
sk = $(\mathbf{x}, \mathbf{y})$
pk = $(\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$
```

Algorithm 2 HQC.Encrypt

```
Input: pk, $\mathbf{m}$, $\theta$
Output: $c = (\mathbf{u}, \mathbf{v})$
SampleInit($\theta$)
$\mathbf{r}_1$ = Sample($\mathcal{R}, \omega_r$)
$\mathbf{r}_2$ = Sample($\mathcal{R}, \omega_r$)
$\mathbf{e}$ = Sample($\mathcal{R}, \omega_e$)
$\mathbf{u} = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2$
$\mathbf{v} = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}$
```

Algorithm 3 HQC.Decrypt

```
Input: sk $= (\mathbf{x}, \mathbf{y})$, $c = (\mathbf{u}, \mathbf{v})$
Output: $\mathbf{m}$
$\mathbf{m} = \mathcal{C}$.Decode$(\mathbf{v} - \mathbf{u} \cdot \mathbf{y})$
```

function in Algorithm 3 is to decode the term $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ which results in

$$\begin{aligned} &(\mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}) - (\mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2) \cdot \mathbf{y} \\ =\ &\mathbf{mG} + (\mathbf{x} + \mathbf{h} \cdot \mathbf{y}) \cdot \mathbf{r}_2 - (\mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2) \cdot \mathbf{y} + \mathbf{e} \\ =\ &\mathbf{mG} + \mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e}. \end{aligned}$$

Thus, the decoder has to correct the error

$$\mathbf{e}' = \mathbf{x} \cdot \mathbf{r}_2 - \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e}.$$

The decoding succeeds if $|\mathbf{e}'| \leq \delta$. The Decryption Failure Rate (DFR) denotes the probability when the weight exceeds the decoder's capacity.

HQC.KEM. The authors of HQC decided to use the Hofheinz-Hövelmanns-Kiltz (HHK) transformation [HHK17] to obtain an IND-CCA secure Key Encapsulation Mechanism from the IND-CPA secure PKE scheme described before. In contrast to the original FO transformation, the HHK approach is able to handle decryption failures. The KEM scheme may be used to share securely a random symmetric key K between two parties. The key generation is the same as for the PKE. The sender of a message applies the encapsulation function in Algorithm 4 to wrap a randomly chosen K and the receiver executes the decapsulation function in Algorithm 5 to obtain the same key or aborts if a decryption failure occurs.

The KEM construction requires the three independent cryptographic hash functions G, K, and H. To encapsulate a randomly chosen message $\mathbf{m}$ the randomness θ for the encryption is derived by $\mathsf{G}(\mathbf{m})$. The shared key K is a linkage of both the message $\mathbf{m}$ and the ciphertext c and is computed by $\mathsf{K}(\mathbf{m}, c)$. Finally, d is derived by computing the hash $\mathsf{H}(\mathbf{m})$.

In the decapsulation, the decryption function is invoked with the secret key sk and the ciphertext c to obtain the message $\mathbf{m}'$. To verify the ciphertext for integrity, a re-encryption of the message $\mathbf{m}'$ is performed using the randomness θ'

Algorithm 4 HQC.Encaps

```
Input: pk
Output: $K$, $(c, d)$
$\mathbf{m} = \mathsf{Sample}(\mathbb{F}_2^{n_1 n_2})$
$\theta = \mathsf{G}(\mathbf{m})$
$c = \mathsf{HQC.Encrypt}(\mathsf{pk}, \mathbf{m}; \theta)$
$K = \mathsf{K}(\mathbf{m}, c)$
$d = \mathsf{H}(\mathbf{m})$
```

Algorithm 5 HQC.Decaps

```
Input: $\mathsf{sk} = (\mathbf{x}, \mathbf{y})$, $(c = (\mathbf{u}, \mathbf{v}), d)$
Output: $K$
$\mathbf{m}' = \mathsf{HQC.Decrypt}(\mathsf{sk}, c)$
$\theta' = \mathsf{G}(\mathbf{m}')$
$c' = \mathsf{HQC.Encrypt}(\mathsf{pk}, \mathbf{m}'; \theta')$
if $c \neq c' \vee d \neq \mathsf{H}(\mathbf{m}')$ then
    $K = \perp$
else
    $K = \mathsf{K}(\mathbf{m}', c)$
end if
```

derived from $\mathbf{m}'$. Then, the procedure checks whether the re-encrypted ciphertext c' matches the received c and whether the sent digest d equals the hash value of the decrypted message $\mathbf{m}'$. If this check succeeds, $\mathsf{K}(\mathbf{m}, c)$ is output, otherwise failure.

2.3 Bit Flipping Key Encapsulation – BIKE

BIKE is another code-based post-quantum KEM targeting IND-CCA security, which is also an alternate candidate in the third round of the NIST PQC competition. As other candidates, it has updated its specification from round to round and we consider the specification submitted to the most recent round of the NIST PQC competition (being round 3) [Ara+21]. It can briefly be considered as the McEliece scheme instantiated with QC-MDPC codes [Mis+13], using the equivalent Niederreiter scheme. Quasi-Cyclic Moderate Density Parity-Check codes are similar to more well known Quasi-Cyclic Low Density Parity-Check (QC-LDPC) codes, but parity checks have a somewhat larger weight ($O(n)$ instead of a constant, where n is the code length). The security of the scheme relies on quasi-cyclic variants of hard decoding problems from coding theory in the Hamming metric. The security level it provides is bounded by the complexity of solving these hard problems with the best known algorithms. In the case of BIKE we have a small probability of decryption failure, which gives on occurrence an error in decapsulation. Typically, a high security level demands that this probability of failure is negligible, say 2^{-128} or even smaller.

Let us now give a brief overview of the specification of BIKE. It is specified from three main values, being the Hamming weight of the error vector t, the row weight of the secret parity check matrix w, and the block length r. To achieve a given security level λ for IND-CPA security, the parameters t and w should be chosen according to the complexity of solving the underlying hard problems. To additionally achieve IND-CCA security, one need to make sure that the decryption

Table 2: BIKE parameters.

Security	r	w	t	DFR
Level 1	12,323	142	134	2^{-128}
Level 3	24,659	206	199	2^{-192}
Level 5	40,973	274	264	2^{-256}

failure probability is upper bounded by $2^{-\lambda}$. The block length r does not affect the computational hardness of the underlying problems much, but do affect the decryption failure probability.

In setup, one sets the target security level λ and then generates the parameters (r, t, w) and an additional parameter l, which gives the size of the shared key output, in bits. We also fix hash functions $\mathsf{H}, \mathsf{K}, \mathsf{L}$ and a decoder Decode. The message space is $\mathcal{M} = \{0,1\}^l$ and the shared key space is $\mathcal{K} = \{0,1\}^l$. We return to the hash functions later. BIKE uses the cyclic polynomial ring $\mathcal{R} = \mathbb{F}_2/(X^r - 1)$; $\mathcal{H}_w$ is the secret key space $\mathcal{H}_w = \{(\mathbf{h}_0, \mathbf{h}_1) \in \mathcal{R}^2 : |\mathbf{h}_0| = |\mathbf{h}_1| = w/2\}$; finally $\mathcal{E}_t$ is the set of errors $\mathcal{E}_t = \{(\mathbf{e}_0, \mathbf{e}_1) \in \mathcal{R}^2 : |\mathbf{e}_0| + |\mathbf{e}_1| = t\}$.

The BIKE KEM consists of several algorithms. First, the key generation KeyGen is done as described in Algorithm 6. It creates the secret key sk, consisting of two low weight vectors $\mathbf{h}_0$ and $\mathbf{h}_1$ of length r, as well as a special value σ, used in case of error in decoding. It also creates the public key pk, being a length r vector computed as $\mathbf{h} = \mathbf{h}_1 \mathbf{h}_0{}^{-1}$. The notation $\mathsf{Sample}(\mathcal{X})$ means that we uniformly pick an element from the set $\mathcal{X}$.

Next, the encapsulation algorithmic Encaps outputs a ciphertext c that contains an encapsulated key value, using only the public key. It first selects a random bitstring m of length l. It then hashes this value to a weight t error vector $(\mathbf{e}_0, \mathbf{e}_1) \in \mathcal{E}_t$. Hashing is done using the special hash function H, which outputs weight t vectors. A ciphertext is then formed, where the first part is $\mathbf{c}_0 = \mathbf{e}_0 + \mathbf{e}_1 \mathbf{h}$. The second part of the ciphertext is $c_1 = m \oplus \mathsf{L}(\mathbf{e}_0, \mathbf{e}_1)$. We note that with knowledge of $(\mathbf{h}_0, \mathbf{h}_1)$ in the secret key, one can efficiently reconstruct $(\mathbf{e}_0, \mathbf{e}_1)$ from $\mathbf{c}_0$. Then one can also reconstruct m from c_1. The final step computes a shared key through $K = \mathsf{K}(m, c)$. All steps are illustrated in Algorithm 7.

The decapsulation algorithmic Decaps is the final algorithmic to describe. It outputs the shared key from the ciphertext c using the secret key. It first computes the error vector used to create $\mathbf{c}_0$ by $\mathbf{e}' = \mathsf{Decode}(\mathbf{c}_0 \mathbf{h}_0, \mathbf{h}_0, \mathbf{h}_1)$. Here Decode is a kind of bit-flipping decoder [Gal62]. The choice of decoder is a trade-off between efficiency and failure probability. In the BIKE specification, the Black-Gray-Flip (BGF) decoder is selected. Next $m' = c_1 \oplus \mathsf{L}(\mathbf{e}')$. If $\mathbf{e}'$ was correctly received, then $m' = m$. This is now checked by computing and comparing if $\mathbf{e}' = \mathsf{H}(m')$. If so, the shared key is set to $K = \mathsf{K}(m, c)$. The steps are illustrated in Algorithm 8.

Algorithm 6 BIKE.KeyGen

```
Input: ·
Output: sk = $(\mathbf{h}_0, \mathbf{h}_1, \sigma)$
    pk = $\mathbf{h} \in \mathcal{R}$
$(\mathbf{h}_0, \mathbf{h}_1) = \mathsf{Sample}(\mathcal{H}_w)$
$\mathbf{h} = \mathbf{h}_1 \mathbf{h}_0^{-1}$
$\sigma = \mathsf{Sample}(\mathcal{M})$
$\mathsf{sk} = (\mathbf{h}_0, \mathbf{h}_1, \sigma)$
$\mathsf{pk} = \mathbf{h}$
```

Algorithm 7 BIKE.Encaps

```
Input: pk = $\mathbf{h}$
Output: $K, c$
$m = \mathsf{Sample}(\mathcal{M})$
$(\mathbf{e}_0, \mathbf{e}_1) = \mathsf{H}(m)$
$c = (\mathbf{e}_0 + \mathbf{e}_1 \mathbf{h}, m \oplus \mathsf{L}(\mathbf{e}_0, \mathbf{e}_1))$
$K = \mathsf{K}(m, c)$
```

Algorithm 8 BIKE.Decaps

```
Input: sk = $(\mathbf{h}_0, \mathbf{h}_1, \sigma)$, $c = (\mathbf{c}_0, c_1)$
Output: $K$
$\mathbf{e}' = \mathsf{Decode}(\mathbf{c}_0\mathbf{h}_0, \mathbf{h}_0, \mathbf{h}_1)$
$m' = c_1 \oplus \mathsf{L}(\mathbf{e}')$
if $\mathbf{e}' = \mathsf{H}(m')$ then
    $K = \mathsf{K}(m', c)$
else
    $K = \mathsf{K}(\sigma, c)$
end if
```

3 Timing Attacks on HQC and BIKE

In this section we present the timing attacks on the schemes HQC and BIKE and the underlying vulnerabilities in both cases.

3.1 The Timing Attack on HQC

In the following, we show how the current optimized HQC implementation [Agu+] from June 2021 which is specified in [Agu+21] leaks timing information which enables the construction of a plaintext distinguisher. Then, this distinguisher is used as a plaintext-checking oracle within existing attacks described in [Bae+19] to achieve the key-recovery on the, now, deprecated version of HQC using BCH and repetition codes. Further, we propose an attack that enables the key-recovery on the current version using Reed-Solomon (RS) and Reed-Muller (RM) codes.

The vulnerability in the HQC implementations. As described in Section 2.2, the encryption function described in Algorithm 2 requires to sample bit vectors of a specified Hamming weight ω. The implementation of the sampling function uses rejection sampling to comply to the security properties, e.g., if a position is sampled twice. The runtime of the rejection sampling algorithm depends on the given seed θ. In the KEM version the en- and decapsulation procedures derive the seed for the encryption function from the message $\mathbf{m}$ by $\mathsf{G}(\mathbf{m})$. The dependence on the message in the decapsulation allows us to construct a plaintext distinguisher which we use to mount a timing attack afterwards.

The Sample function. The considered implementation of HQC implements the weighted vector sampling in the function `vect_set_random_fixed_weight`. For brevity we refer to this function as Sample. In each iteration the function generates random positions from the range $\{0, \ldots, n-1\}$ to set a bit at that

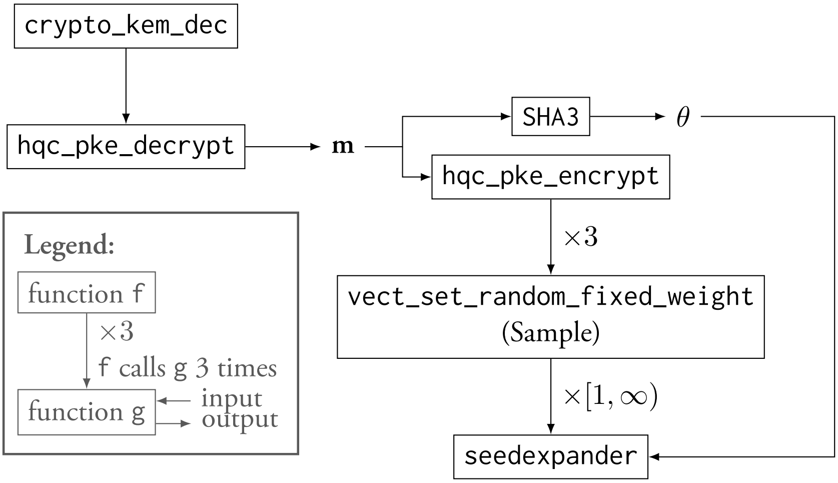


Figure 1: Visualization of the information flow in the decapsulation function of the current HQC KEM implementation [Agu+].

position to 1 until w distinct bit positions have been sampled. Concretely, if the sampled bit position has already been sampled before, the sample is rejected. Otherwise, the bit position is stored in an array. At the end, the vector of weight w is constructed by setting the bits at the w distinct positions that were sampled. The number of times a bit position collides with a previously sampled bit position is directly proportional to the runtime of the algorithm.

The randomness in the Sample function is deterministic and determined by an eXtendable-Output Function (XOF) implemented by the seedexpander function. For our analyses we assume that the outputs of the XOF are uniformly, independent and identically distributed (iid). The XOF influences the path that is taken through the function and is initialized with the seed $\theta = \mathsf{G}(\mathbf{m})$. The message $\mathbf{m}$ is obtained from the decoding of the ciphertext c, c.f., ?? 274 in Algorithm 3. This data flow is illustrated in Fig. 1. Therefore, the message $\mathbf{m}$ controls how many iterations the rejection sampling algorithm takes. Further, a rejection leads to another call of the seedexpander function and, thus, to a large timing gap.

Additional seedexpander calls. We refer to seedexpander calls which are executed conditionally within the loop in the Sample function, c.f. Fig. 1, as *additional* seedexpander *calls*. For details, we refer to the original source code which can be found in the file vector.c, line 31, in [Agu+]. In general, unless otherwise specified, we only count the number of additional seedexpander calls and skip the default initial call. The seedexpander is initially used to produce $3 \cdot \omega_r$ bytes of randomness and store it into a buffer. If this randomness is sufficient to generate ω_r distinct bit positions, no additional seedexpander calls are issued. However, if even a single sample is rejected the algorithm will need to produce additional randomness by issuing another `seedexpander` call. The sampled bit positions are in the range of $\{0, \ldots, n-1\}$. To generate these positions, the algorithm performs an inner rejection sampling algorithm. The inner rejection sampling algorithm samples a position from $\{0, \ldots, 2^{24}-1\}$ that is to be reduced modulo n, where $n < 2^{24}$. However, the position is rejected if it is above the largest multiple of n that is smaller than 2^{24} which is defined by $\eta := \lceil 2^{24}/n \rceil\, n$ or `UTILS_REJECTION_THRESHOLD` in the implementation. This is to avoid biasing the distribution and discussed in detail in Section 5.2.

Thus, sampling distinct bit positions can fail in two ways: (1) The sampled position in $\{0, \ldots, 2^{24}-1\}$ is larger than η or (2) it collides with a previously sampled one. We can model rejection sampling of a position as a Bernoulli variable with the success probability $p = \eta/2^{24}$. Each attempt to generate a valid bit position below n consumes 3 bytes of randomness. If the algorithm succeeds in picking a distinct bit position in every iteration, it does not need additional randomness. In this case `seedexpander` is not called within the for loop. However, if even a single sample fails or collides the algorithm will need to produce additional randomness, as it now requires more than $3 \cdot \omega_r$ bytes. The probability of all ω_r samples succeeding and picking distinct positions out of n bit positions is

$$\tilde{p} = \prod_{i=0}^{\omega_r - 1} \left(p \frac{n-i}{n} \right)$$

which evaluates, for instance, to approx. 81.95% for the hqc-128 parameter set. Thus, only $1 - \tilde{p} \approx 18.05\%$ of all possible seeds θ result in at least one additional call to the `seedexpander` function. The probabilities for all parameter sets are listed in Table 3.

Decapsulation timing. Inspecting the decapsulation function in Algorithm 5 the timing variation is caused by the invocation of the encryption function using the seed $\theta = \mathsf{G}(\mathbf{m})$. Viewing the encryption function in Algorithm 2 we observe three calls to the previously discussed `Sample` function. One for each of the random vectors: $\mathbf{r}_1, \mathbf{r}_2, \mathbf{e}$, where the weight parameters ω_r and ω_e are equal. Each of these calls is using the same `seedexpander` instance, whose randomness depends upon the seed θ. In each of these three invocations there is a $1 - \tilde{p}$ chance that `seedexpander` is called at least once within the for loop. Thus, $(1 - \tilde{p})^3$ of messages result in three or more calls to `seedexpander`.

The Distinguisher

Given a ciphertext c we can distinguish whether the decrypted message $\mathbf{m}$ yields the same timing behavior during the encryption as another ciphertext. We define

Table 3: The approximated probabilities p for successfully sampling a bit position in the range required for unbiased modulo reduction, $\tilde{p}$ for completing the rejection sampling routine without exhausting the initially generated randomness, and for a message that causes at least 3 additional `seedexpander` invocations.

Instance	p (in %)	$\tilde{p}$ (in %)	$(1-\tilde{p})^3$ (in %)
hqc-128	99.94	81.95	0.58
hqc-192	99.79	65.93	3.95
hqc-256	99.97	79.09	0.91

a distinguisher $\mathcal{D}$ as:

$$\mathcal{D}^{\mathcal{O}}(c_1, c_2) := \mathcal{O}(c_1) \stackrel{?}{=} \mathcal{O}(c_2) \tag{1}$$

where $\mathcal{O} = \mathit{TB}(\mathsf{sk}, \cdot)$ is the decapsulation timing oracle and yields the timing behavior – the number of `seedexpander` calls – of the provided ciphertext under the secret key sk and $\cdot \stackrel{?}{=} \cdot$ returns whether the two arguments are equal or not. The advantage of $\mathcal{D}$ when distinguishing a given ciphertext c_1 that decrypts to $\mathbf{m}_1$ from another ciphertext c_2 that decrypts to a uniform randomly chosen message $\mathbf{m}_2$ is given by:

$$\begin{aligned}
&|\Pr_{c_2 \xleftarrow{\$} \mathcal{C}}[\mathcal{D}^{\mathit{TB}(\mathsf{sk},\cdot)}(c_1, c_2) = 1 \mid \mathsf{Decrypt}(\mathsf{sk}, c_1) = \mathsf{Decrypt}(\mathsf{sk}, c_2)] - \\
&\quad \Pr_{c_2 \xleftarrow{\$} \mathcal{C}}[\mathcal{D}^{\mathit{TB}(\mathsf{sk},\cdot)}(c_1, c_2) = 1 \mid \mathsf{Decrypt}(\mathsf{sk}, c_1) \neq \mathsf{Decrypt}(\mathsf{sk}, c_2)]| \\
= &|\Pr_{c_2 \xleftarrow{\$} \mathcal{C}}[\mathit{TB}(\mathsf{sk}, c_1) = \mathit{TB}(\mathsf{sk}, c_2) \mid \mathsf{Decrypt}(\mathsf{sk}, c_1) = \mathsf{Decrypt}(\mathsf{sk}, c_2)] - \\
&\quad \Pr_{c_2 \xleftarrow{\$} \mathcal{C}}[\mathit{TB}(\mathsf{sk}, c_1) = \mathit{TB}(\mathsf{sk}, c_2) \mid \mathsf{Decrypt}(\mathsf{sk}, c_1) \neq \mathsf{Decrypt}(\mathsf{sk}, c_2)]| \\
= &\ 1 - \Pr_{c_2 \xleftarrow{\$} \mathcal{C}}[\mathit{TB}(\mathsf{sk}, c_1) = \mathit{TB}(\mathsf{sk}, c_2) \mid \mathsf{Decrypt}(\mathsf{sk}, c_1) \neq \mathsf{Decrypt}(\mathsf{sk}, c_2)]
\end{aligned}$$

where $\mathcal{C}$ is the ciphertext space. The last formula shows that the advantage is at a maximum when the probability of obtaining the same timing behavior for another ciphertext c_2 that decrypts to a different message is at a minimum. We can achieve this by minimizing the probability of the timing behavior of c_1 by picking a suitable message $\mathbf{m}_1$.

The Secret Key Recovery Attack

By using the observations of the vulnerability analysis to get a distinguisher described in Section 3.1 for a secret key recovery we propose the following attack idea. We pick a message $\mathbf{m}$ that has the property of resulting in 3 additional calls to the `seedexpander` function. Regarding the low probabilities in Table 3, we know that most of the messages do not share this property with our chosen message $\mathbf{m}$. Therefore, since we can determine whether a decryption has resulted in exactly 3 calls or not through the timing behavior, we can distinguish whether a ciphertext decrypts to the message $\mathbf{m}$ with high advantage. Next, we compute a ciphertext $c = (\mathbf{u}, \mathbf{v})$ by manually setting $\mathbf{r}_1$ to $1 \in \mathcal{R}$, and $\mathbf{r}_2$ and $\mathbf{e}$ to $0 \in \mathcal{R}$ during the encryption of $\mathbf{m}$. This ciphertext has the desirable property, that the error that the decoder has to correct during the decryption is just $\mathbf{y}$, a part of the secret key:

$$\mathbf{v} - \mathbf{u} \cdot \mathbf{y} = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e} - (\mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2) \cdot \mathbf{y} = \mathbf{mG} - \mathbf{r}_1 \cdot \mathbf{y} = \mathbf{mG} - \mathbf{y}. \quad (2)$$

If we are able to find the error $-\mathbf{y} = \mathbf{y}$, we can compute the remaining part of the secret key as $\mathbf{x} = \mathbf{s} - \mathbf{h} \cdot \mathbf{y}$. Note, that we do not need $\mathbf{x}$ as it is never used during the decapsulation. Further, note that this ciphertext is not valid, since we cannot fully control $\mathbf{r}_1, \mathbf{r}_2$, or $\mathbf{e}$ during the encryption. For valid ciphertexts, these are derived from $\mathbf{m}$ via the XOF and the `Sample` function. We do not require a valid ciphertext, as our timing-side channel will reveal information, even if the ciphertext is rejected by the decapsulation oracle.

To recover the error $\mathbf{y}$ we follow the basic principles outlined by Hall et al. [HGS99]. The authors propose adding an error $\mathbf{e}'$ to the ciphertext c until we detect that the modified ciphertext c' decrypts to a different message $\mathbf{m}'$. Then, we test for every bit b in the ciphertext c', whether flipping it causes the ciphertext to decrypt back to the original message $\mathbf{m}$. If it does, we know that the bit b is an error bit in the modified ciphertext c'. Otherwise, b is not an error.

Unfortunately, we cannot directly apply this method to HQC for several reasons: (1) Instead of correcting errors we need to determine the error e of our original ciphertext $c = \mathbf{mG} + \mathbf{e}$. (2) Further, when flipping erroneous bits in the modified ciphertext it does not decrypt back to the original message in most cases. Thus, we would not detect that the bit is an error. (3) Finally, the timing side-channel can not distinguish pairs of messages that induce the same number of `seedexpander` calls. Therefore, we sometimes do not detect that our modified ciphertext does not decrypt to the same message $\mathbf{m}$ anymore.

The first issue can be solved by keeping track of the error $\mathbf{e}'$ that we add to $c = \mathbf{mG} + \mathbf{e}$ to obtain $c' = c + \mathbf{e}'$. If we flip a bit b in $\mathbf{e}'$ to obtain the ciphertext c' and it decrypts back to the original message $\mathbf{m}$, we know that b is an error in $c' = c + \mathbf{e}'$. Let $\mathbf{e}'' = \mathbf{e} + \mathbf{e}'$. If the bit b is set in $\mathbf{e}''$, then b is set in $\mathbf{e}$ if and only if the b-th bit of $\mathbf{e}'$ is not set. Or in other words, if we did not introduce the error ourselves, we know that the bit is an error. Otherwise, we know that the bit is correct. The second issue vastly increases the number of timing oracle calls since it introduces a very high false negative rate. We do not gain any information if the ciphertext does not decrypt back to the original message. To address this issue, we retry the entire function multiple times, with many different $\mathbf{e}'$. Eventually, we obtain a decision for every bit. The third issue may be solved by obtaining three or more decisions for every bit, and then obtaining a final decision with a majority vote.

Our resulting attack approach is detailed in Algorithm 9. First, we need to find a proper message $\mathbf{m}$ which yields 3 additional `seedexpander` calls. Therefore, we perform an exhaustive but low effort search. According to Eq. (2), we apply the modified encryption to $\mathbf{m}$ to obtain the initial ciphertext $c = (\mathbf{u}, \mathbf{v})$. Further, we define a proper majority threshold T as the majority of N votes. Afterwards, we apply Algorithm 10 to find another ciphertext $c' = (\mathbf{u}, \mathbf{y}+\mathbf{e}')$ and the corresponding $\mathbf{m}'$ that differs from $\mathbf{m}$. We only add $\mathbf{e}'$ to $\mathbf{v}$ because the input to the decoder evaluates to additional errors just in the secret key part $\mathbf{y}$, c.f., Eq. (3).

$$\text{Decrypt}(\text{sk}, (\mathbf{u}, \mathbf{v}+\mathbf{e}')) = \mathcal{C}.\mathsf{Decode}(\mathbf{v}+\mathbf{e}' - \mathbf{u}\cdot\mathbf{y}) = \mathcal{C}.\mathsf{Decode}(\mathbf{mG}+\mathbf{e}' - \mathbf{y}) \tag{3}$$

In particular c' should have exactly one more error bit than the decoder could correct. From this state, flipping any bit in c' and checking whether the ciphertext decodes again reveals whether that bit was an error bit in c or not. We can exploit this property to recover $\mathbf{y}$ later on. Starting from c and an error of $\mathbf{e}' = 0$, we iteratively increase the weight of $\mathbf{e}'$ by flipping single, random bits. After each flip, we send the modified ciphertext to the decapsulation timing oracle $\mathcal{D}^{\mathrm{TB}(\mathsf{sk},\cdot)}$ and check if the ciphertext causes a different amount of time in the decryption operation than our original ciphertext. If it does, we have found a ciphertext c' that decrypts to a different message $\mathbf{m}'$.

Then, for each bit position b in $\mathbf{v}+\mathbf{e}'$, we flip the bit and send $(\mathbf{u}, \mathbf{v}+\mathbf{e}'+2^b)$ to the decapsulation timing oracle, where 2^b is a vector with the b^{th} bit set. If we detect that the timing is again equal to the timing of our original ciphertext, we assume that the decryption yields back the original message $\mathbf{m}$ and that the corresponding bit in the secret key part $\mathbf{y}$ is set. Otherwise, we assume that the ciphertext decrypts to a different message and that there is no error bit set at this position.

Finally, Algorithm 9 calls Algorithms 10 and 11 multiple times until a majority is revealed at each bit position for a 0- or 1-bit. To determine the majorities the counters in t record the total number of votes that have been cast for each bit b. The counters in r record the number of 1-votes for each bit b, i.e., the number of votes that the bit b is set. The number of 0-votes for a bit b is computed by $\mathsf{t}[b] - \mathsf{r}[b]$. For a majority either the number of 1-votes or the number of 0-votes has to exceed $\lfloor N/2 \rfloor + 1$.

Reducing the number of oracle queries. We can improve the attack by targeting a specific word of the duplicated RM code. Specifically, consider that the code used in HQC is a concatenated code combining an outer RS code with an inner duplicated RM code. During encoding, each element in the alphabet $\mathbb{F}_q$ from a

Algorithm 9 KeyRecovery

```
Input: Ciphertext $c$, parameter $N$ to compute majority threshold $T$.
Output: y.
for $b = 0$ to $n_1 n_2 - 1$ do
    $\mathsf{y}[b] \leftarrow 0, \mathsf{t}[b] \leftarrow 0, \mathsf{r}[b] \leftarrow 0$
end for
repeat
    $(c', \mathsf{bs}) \leftarrow \mathsf{FindDiffMsg}(c)$
    $\mathbf{e}' \leftarrow \mathsf{RecoverError}(c')$
    majority← true
    $T \leftarrow \lceil \frac{N}{2} \rceil + 1$
    for $b = 0$ to $n_1 n_2 - 1$ do
        if $\mathbf{e}'[b] = 1$ then
            $\mathsf{t}[b] \leftarrow \mathsf{t}[b] + 1$
            if $b \notin \mathsf{bs}$ then
                $\mathsf{r}[b] \leftarrow \mathsf{r}[b] + 1$
            end if
        end if
        if $\mathsf{r}[b] < T$ and $\mathsf{t}[b] - \mathsf{r}[b] < T$ then
            majority← false
        end if
    end for
until majority= true
for $b = 0$ to $n_1 n_2 - 1$ do
    $\mathsf{y}[b] \leftarrow \mathsf{r}[b] \geq T$
end for
return e
```

word of the outer code is mapped to a message that the inner code can encode. We can obtain an oracle whether a word of the inner code decoded correctly by corrupting **v** such that a single additional corrupted inner code word would result in a decoding failure. We achieve this by corrupting δ – the error correction capacity of the outer code – elements of the outer code. We then may add an error $\mathbf{e}'$ to a single element of words of the RS code. A similar procedure has been previously described [Bae+19, Ex.15] to attack Lepton [YZ17] which uses BCH and repetition codes.

The oracle we construct here may also enable faster attacks [Waf+19] if the noise learning problem [Bae+19] is solved for duplicated RM codes. We do not implement such a version of the attack as we are not aware of a solution to this problem.

Algorithm 10 FindDiffMsg

```
Input: $c$
Output: $c'$, flipped bits $\mathsf{bs}$
$c' \leftarrow c$
$\mathsf{bs} \leftarrow \mathsf{RandomPermutation}([0, \ldots, n_1 n_2 - 1])$
for $i = 0$ to $n_1 n_2 - 1$ do
    Flip bit $\mathsf{bs}[i]$ in $\mathbf{v}$ of $c'$
    if $\mathcal{D}^{\mathrm{TB}(\mathsf{sk},\cdot)}(c, c') = 0$ then
        return $(c',\ \mathsf{bs}[0, \ldots, i])$
    end if
end for
```

Algorithm 11 RecoverError

```
Input: Modified ciphertext $c'$
Output: Combined error $\mathbf{e}$
$\mathbf{e} \leftarrow \mathbf{0}$
for $i = 0$ to $n_1 n_2 - 1$ do
    Flip bit $i$ in $\mathbf{v}$ of $c'$
    if $\mathcal{D}^{\mathrm{TB}(\mathsf{sk},\cdot)}(c, c') = 1$ then
        Set bit $i$ in $\mathbf{e}$
    end if
    Flip bit $i$ in $\mathbf{v}$ of $c'$
end for
```

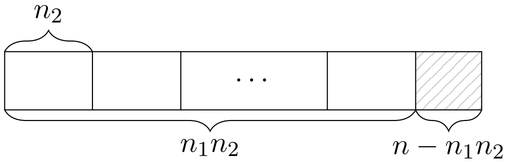


Figure 2: An element of $\mathbb{F}_2[x]/\langle x^n - 1 \rangle$ and its segmentation into codewords of the inner code.

Recovering the entire secret key. Using the methods described so far we can recover $n_1 n_2$ bits of the secret key $\mathbf{y}$. However, we are missing $n - n_1 n_2$ bits, that are required for using y during decryption. In Fig. 2 the structure of HQC codewords is displayed. Depending on the codes used, there are n_1 RM or repetition code codewords. However, $n - n_1 n_2$ bits of the n bits in total are never used during decoding. Thus, these bits cannot be obtained using the methods described so far. We now show how this situation can be remediated, and how it does not have a significant impact on the success probability, when the attack accounts for it. This issue was not addressed in some other attacks against HQC [Waf+19]. Fortunately, the difference between n and $n_1 n_2$ is small for most parameters. However, for some parameters the difference could dominate the attack's complexity, if we were to brute force every possible combination. The largest difference with the new parameter sets is 37 bits in hqc-256. We can check whether a combination of bits is correct by checking whether we can decrypt an honestly encrypted message successfully. Fortunately, we can drastically reduce the search space while retaining a very high success probability. Assuming the number of bits set in the remaining bits is ≤ 2, the number of ways to pick these bits is $\sum_{i=0}^{2} \binom{n - n_1 n_2}{i}$. This number is low enough for all parameter choices to enumerate using a brute force search.

Table 4: Remaining $n - n_1 n_2$ bits that must be recovered for each parameter set, the number of ways to pick the remaining bits with a weight of up to 2, the probability that the weight is 0, and the probability that the weight is ≤ 2.

Instance	$n_1 n_2$	n	ω	$n - n_1 n_2$	$\sum_{i=0}^{2} \binom{n-n_1 n_2}{i}$	Pr$[Z = 0]$	Pr$[Z \leq 2]$
hqc-128	17,664	17,669	66	5	16	$\approx 98.1\,\%$	$\approx 100.0\,\%$
hqc-192	35,840	35,851	100	11	67	$\approx 97.0\,\%$	$\approx 100.0\,\%$
hqc-256	57,600	57,637	131	37	704	$\approx 91.9\,\%$	$\approx 100.0\,\%$

We now investigate the success probability given this dramatic search space reduction. We define $Y_{i,o,w}$ to be the number of elements that land inside a region of i elements when sampling w distinct elements uniformly from a region of $i + o$ elements. The region i (or "inside") corresponds to the bits that are set in the remaining $n - n_1 n_2$ bits. The region o (or "outside") corresponds to the $n_1 n_2$ bits that we have already obtained using the attack. Then the probability that x of the w distinct elements land inside the region of i elements is:

$$\Pr[Y_{i,o,w} = x] = \frac{\binom{i}{x}\binom{o}{w-x}}{\binom{o+i}{w}}$$

We now let $Z = Y_{n-n_1 n_2, n_1 n_2, \omega}$. Assuming the attack was successful for all $n_1 n_2$ bits, the success probability is approx. 98.1% for hqc-128 when we guess that all remaining bits are zero, represented by the column Pr$[Z = 0]$ in Table 4. However, this loss is preventable by brute-forcing the remaining bits. We can come very close to a success probability of 1, even for a modest search of only ≤ 2 set bits.

3.2 The Timing Attack on BIKE

Central elements for our attack are the decoding algorithm and the hash functions, which are described here a bit more. From the specification we see that the decoding step calls $\mathsf{Decode}(\mathbf{s}, \mathbf{h}_0, \mathbf{h}_1)$ which returns either $(\mathbf{e}_0, \mathbf{e}_1) \in \mathcal{R}^2$ such that $\mathbf{e}_0\mathbf{h}_0 + \mathbf{e}_1\mathbf{h}_1 = \mathbf{s}$ or the failure symbol $\perp$. First, note that there is no restriction on the weight of the returned error in the Decode algorithm. Any weight is possible as long as $\mathbf{e}_0\mathbf{h}_0 + \mathbf{e}_1\mathbf{h}_1 = \mathbf{s}$. Secondly, if decoding is not successful and the failure symbol is returned, it has to be coded into a binary value. In existing reference implementations, failure is indicated by assigning a specific value like $(\mathbf{e}_0, \mathbf{e}_1) = 0$.

For the hash functions used, K and L are considered as standard hash functions, mapping to l-bit strings. But H is a special hash function, since its output is a vector of weight t. It finds its output by a rejection sampling method. Its description is given in Algorithm 12[4] and uses also a pseudorandom number generator called

[4] Compared to Algorithm 3 in the BIKE specification, we fixed line 7 and add a new line for the

Algorithm 12 WAES-CTR-PRF

```
Input: seed, $w$ (32 bits), len
Output: A list of $w$ different bit-positions in $\{0, ..., \text{len} - 1\}$.
 1: wlist = $\phi$; ctr = 0; $i = 0$
 2: $s$ = AES-CTR-Stream(seed, $\infty$)          ▷ $\infty$ denotes "sufficiently large"
 3: mask = $(2^{\lceil \log_2 r \rceil} - 1)$
 4: while ctr < w do
 5:     pos = $s[32(i+1) - 1 : 32i]$ & mask      ▷ & denotes bitwise AND
 6:     if ((pos < len) AND (pos $\notin$ wlist)) then
 7:         wlist = wlist $\cup$ pos
 8:         ctr = ctr + 1
 9:     end if
10:     $i = i + 1$
11: end while
12: return wlist, s
```

AES-CTR-Stream$(\cdot)$[5] in the round-3 submission to NIST. In brief, the algorithm is producing a list of w different bit-positions in $\{0, ..., len-1\}$, which correspond to the positions of the ones in the weight $t = w$ error vector of length $len = 2n$ that should be the output of the H hash function. The first step in the algorithm is to call AES-CTR-Stream$(\cdot)$ to get a new position value and add it to the list if it was not previously already selected. The number of required calls for randomness (the final value of the i variable) varies, depending on the number of collisions with already selected values.

The situation in BIKE is very similar to the HQC case. Looking at the definition of Encaps/Decaps (see Algorithms 7 and 8) we have seen that the rejection sampling takes place in the H function, in order to generate a random error vector of fixed-weight. A non-constant time implementation of H thus means that we can distinguish between the cases $m = m'$ and $m \neq m'$ with some probability. The value m' directly depends on the ability of the decoder to correctly extract the error vector $e = (\mathbf{e}_0, \mathbf{e}_1)$ from the ciphertext c. This means that we have, as for the case of HQC, a distinguisher between chosen ciphertexts above and below the error correction capability of the decoder. This assumes the rejection sampling algorithm is not implemented in constant time, of course. BIKE officially claims only a IND-CPA secure scheme with the ephemeral key use-case, although they claim IND-CCA security if the decoder they use can be shown to have a decoding failure rate lower than the bit-security level of the scheme.

We are now ready to formulate an attack on BIKE, based on the described observations. As before, we consider an IND-CCA scenario, where we assume

iteratoration of i, so that it is outside of the if-statement, c.f., line 9 in Algorithm 12.

[5] In the most recent version, the designers instead employ SHAKE256.

that we can compute ciphertexts (with encapsulated keys), feed a ciphertext for decapsulation and observe the output of decapsulation. This may for example be an attempt to establish a joint key. As this is a timing attack, we also add the assumption that we get timing information from the decapsulation step.

We leverage the GJS attack [GJS16; NJW18] and use the rejection sampling vulnerability as way to act as a distinguisher of decoding failure. This attack assumes that the scheme is used in a static key setting requiring IND-CCA security. The *Error Amplification* attack [NJW18] builds on the GJS attack [GJS16], but requires only a single initial error vector that results in a decoding failure and then modifies this in order to generate many more error vectors. Let us give very brief descriptions of these attacks.

The GJS Attack

The GJS attack [GJS16; NJW18] was described as an attack on QC-MDPC public-key schemes, using decryption failures that occur. As BIKE is a QC-MDPC scheme, the attacks are directly applicable. In our case, the secret key is $(\mathbf{h}_0, \mathbf{h}_1)$, which also determines the secret parity-check matrix of the code to be decoded. Central is the notion of distance between two ones at position i_1 and i_2, $i_1 < i_2$, in a vector. It is defined as the smallest value of $(i_2 - i_1)$ and $(i_1 - i_2) + r$, where r is the length of the vector (the smallest distance between the two ones in cyclic sense).

The *distance spectrum* for a length r vector x is denoted $D(x)$. It is (in its simplest form) defined as

$$D(x) = \{d : 1 \leq d \leq r/2, d \text{ is a distance existing in } x\}.$$

It can be extended by also introducing $\mu(d)$, where $\mu(d)$ is the number of times the distance d is present in vector x, when $d \in D(x)$.

The approach is now to examine the decoding result for different error patterns. In particular, one picks errors from special subsets. For example, let Ψ_d be the set of all binary vectors of length $n = 2r$ having exactly t ones, where all ones are placed with distance d in the first half of the vector. The other half of the vector is zero. The construction of Ψ_d gives repeated ones at distance d at least $t/2$ times, where

$$\begin{aligned} \Psi_d = \quad & \{(\mathbf{e}, \mathbf{0}) \mid \exists \text{ distinct } s_1, s_2, \ldots, s_t, \text{ s.t. } \mathbf{e}_{s_i} = 1, \text{ and} \\ & s_{2i} = (s_{2i-1} + d) \bmod r \text{ for } i = 1, \ldots, t/2, \text{ and } |\mathbf{e}| = t\} \end{aligned} \tag{4}$$

In the attack phase one sends many messages with the error selected from the subset Ψ_d. When there is a decoding error one records this. With enough samples one can compute an empirical decoding error probability for the subset Ψ_d. Furthermore, this is done for $d = 1, 2, \ldots, r/2$. The main observation is that there is a strong correlation between the decoding error probability for error vectors from

Ψ_d and the existence of a distance d between two ones in the secret vector $\mathbf{h_0}$. If there exists two ones in $\mathbf{h_0}$ at distance d, the decoding error probability is much smaller than if distance d does not exist between two ones.

After sending many messages, we look at the decoding error probability for each Ψ_d and classify each d, $d = 1, 2, \ldots, U$ according to its multiplicity $\mu(d)$, since each distance can appear not only once but many times. This provides a distance spectrum for the secret vector $\mathbf{h_0}$, which we write $D(\mathbf{h_0})$. Finally, from $D(\mathbf{h_0})$ it is an easy task to compute $\mathbf{h_0}$. One can even have a smaller number of wrong values in $D(\mathbf{h_0})$ and still be able to compute $\mathbf{h_0}$. We list the basic key reconstruction algorithm from [GJS16] in Algorithm 13 for completeness. The advanced version that is capable of recovering keys from distance spectrum with errors is proposed in [GJW19].

Algorithm 13 Key recovery from distance spectrum (from [GJS16])

```
Input: distance spectrum $D(\mathbf{h_0})$, partial secret key $\mathbf{h_0}$, current depth $l$
Output: recovered secret key $\mathbf{h_0}$ or message ”No such secret key exists” Initial
    recursion parameters: distance spectrum $D(\mathbf{h_0})$, empty set for secret key,
    current depth 0
if $l = w$ then
    return $\mathbf{h_0}$                                        ▷ secret key found
end if
for all potential key bits $i$ do
    if all distances to key bit $i$ exist in $D(\mathbf{h_0})$ then
        Add key bit $i$ to secret key $\mathbf{h_0}$
        Make recursive call with parameters $D(\mathbf{h_0})$, $\mathbf{h_0}$ and $l + 1$
        if recursive call finds solution $\mathbf{h_0}$ then
            if $\mathbf{h_0}$ is the secret key then
                return $\mathbf{h_0}$                            ▷ secret key found
            end if
        end if
        Remove key bit $i$ from secret key $\mathbf{h_0}$
    end if
end for
return ”No such secret key exists”
```

The Secret Key Recovery Attack

The attack now follows the procedure listed in Algorithm 14 and using the distinghuisher in Algorithm 15. Let us step through the different parts of the attack in more detail.

Algorithm 14 Pseudo code of attack: $\mathsf{BikeAttack}(h, w^*, M, I)$

```
$m \leftarrow$ Plaintext such that $\mathsf{H}(m)$ is easily distinguishable
by timing attack                                        ▷ Preamble
loop
    Generate random $\mathbf{e}$, of hamming weight $w^*$
    if DecodingFailureDistinguisher($\mathbf{e}, I$) = True then
        break                ▷ Found the first e which causes a decoding failure
    end if
end loop
$F, G, A, B \leftarrow$ empty vectors                    ▷ Main body
for $i \leftarrow 1, M$ do
    $\mathbf{e}^* \leftarrow$ Move random non-zero bit in $\mathbf{e}$
    $\Delta D_d \leftarrow$ Distance spectrum differences between $\mathbf{e}$ and $\mathbf{e}^*$
    if DecodingFailureDistinguisher($\mathbf{e}^*, I$) = True then
        $\mathbf{e} \leftarrow \mathbf{e}^*$
        Update lists $F, G$ with $\Delta D_d$ according to [NJW18]
    else
        Update lists $A, B$ with $\Delta D_d$ according to [NJW18]
    end if
end for
$A' \leftarrow \max(A) - A + \min(A)$                   ▷ Postamble
$D \leftarrow F + G + A' + B$
Recover secret key with distance spectrum $D$ as per [GJS16]
```

1. We start by finding m such that $\mathsf{H}(m)$ is easily distinguishable by a timing attack. Here we pick m with an extraordinarily distinct timing profile (i.e. long or short) in the *rejection sampling*. It means that we have many or few collisions in Algorithm 12 that makes execution require more or less time, than the average case. Selection of strategy, as well as details on the number of calls, will be discussed later.

2. Construct an error pattern $e = (\mathbf{e}_0, \mathbf{e}_1)$ with higher than normal Hamming weight so that we are as close to the decoding limit as possible. We assume one part of e (w.l.o.g., $\mathbf{e}_1$) is an all-zero vector.

3. Calculate c and transmit to target.

4. Determine if $m' \stackrel{?}{=} m$ by timing attack.

5. Repeat from step 2. to collect many e where $m' \neq m$ as per GJS attack or *Error Amplification* attack

Algorithm 15 DecodingFailureDistinguisher($\mathbf{e}, I$)

```
$\mathbf{e}_0, \mathbf{e}_1 \leftarrow \mathbf{e}$
$c \leftarrow (\mathbf{e}_0 + \mathbf{e}_1\mathbf{h}, m \oplus \mathsf{L}(\mathbf{e}_0, \mathbf{e}_1))$
$S \leftarrow \emptyset$
for $i \leftarrow 1, I$ do
    start $\leftarrow$ RDTSCP
    BIKE.Decaps(c)
    stop $\leftarrow$ RDTSCP
    $S \leftarrow S \cup$ (stop $-$ start)
end for
Determine decoding failure $f$ or not with $S$
return $f$
```

6. After sufficiently many errors are collected, we could determine the distance spectrum statistically.

7. The secret key can be recovered via the reconstruction method in [GJS16] or the improved reconstruction method in its extended version [GJW19] that can handle errors in the recovered distance spectrum.

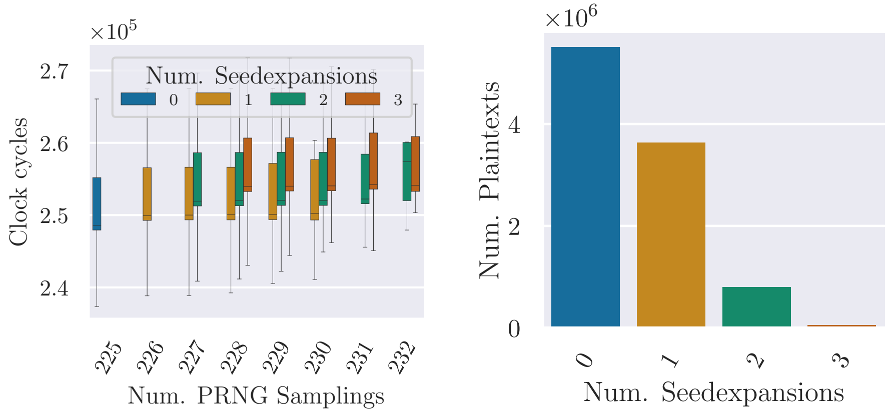
We can check the correctness of this approach. First, L takes input from $\mathcal{R}^2$, so it delivers a result for any choice of $e = (\mathbf{e}_0, \mathbf{e}_1)$. Hence we can build ciphertexts accordingly. In the decaps step, $\mathbf{c}_0 \in \mathcal{R}$ is always a valid input to the decoder. The decoder delivers an error $\mathbf{e}'$ or a failure. But since the result from the decoder is fed into the L with input in $\mathcal{R}^2$, the failure symbol must be interpreted as a fixed value in $\mathcal{R}^2$ (as is also done in the reference code). Altogether, there are no problems with the domain and range of functions. We can feed decaps with ciphertext corresponding to error vectors with higher weight than specified. It is only in the last check of $\mathbf{e}' = \mathsf{H}(m')$ that it will fail, since H is only delivering error vectors of weight t.

4 Evaluation

In this section we present the empirical evaluations of both attack approaches described in Sections 3.1 and 3.2.

4.1 Empirical Evaluation of the Attack on HQC

To confirm the previously postulated hypothesis about the timing behavior we performed a leakage assessment by measuring the CPU cycles of the decapsulation function in the hqc-128 setting for ten million random ciphertexts. Fig. 3a shows how more `seedexpander` calls result in an increased running time. We observe up to 3 additional seedexpander calls. In Fig. 3b we can see the frequency of different timing behaviors. As expected, the frequency decreases when the number of additional seedexpander calls increases. Further, the rate of three additional calls is low enough to be distinguishable to the other three cases. The probability of four additional calls is negligible and does not occur.



(a) For each observed combination of the number of additional seedexpander calls and the number of times a position was attempted to be sampled we show a boxplot of the number of cycles that the decapsulation function took.

(b) Bar plot of the number of additional seedexpander calls observed for the 10 million random ciphertexts generated. 3 additional seedexpander calls corresponds to the rarest observed timing behavior.

Figure 3: Decapsulation timings and frequency of different timing behaviors. We observe that the running time of the decapsulation function is proportional to the number of seedexpansions and that more seedexpander calls are rare. The left figure shows a standard box plot with the median indicated within the box, which also shows the quartiles. The whiskers extend to show 1.5 times the interquartile range.

We have empirically verified the existence of the timing variation by generating random ciphertexts under a single keypair and measuring the number of cycles that the decapsulation algorithm required for 100 random ciphertexts. To measure the number of cycles that an operation takes we use the rdtsc instruction on x86 as recommended by Intel [Pao10]. Section 5.5 shows whether there is a difference in decapsulation time between pairs of 100 ciphertexts generated for a single keypair. We determine whether there is a statistically significant difference using Welch's t-test [Wel47] ($\alpha = 0.1\%$). The t-statistic for two distributions X_1 and X_2 in Welch's t-test is computed as:

$$\frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s^2_{X_1}}{N_1} + \frac{s^2_{X_2}}{N_2}}} \tag{5}$$

where $\bar{X}_i$, $s^2_{X_i}$ and N_i are the sample mean, variance and size of X_i, respectively. The degrees of freedom are estimated by the Welch-Satterthwaite equation:

$$\nu = \frac{\left(\frac{s^2_{X_1}}{N_1} + \frac{s^2_{X_2}}{N_2}\right)^2}{\frac{\left(\frac{s^2_{X_1}}{N_1}\right)^2}{N_1-1} + \frac{\left(\frac{s^2_{X_2}}{N_2}\right)^2}{N_2-1}}. \tag{6}$$

The results show that many pairs of ciphertexts emit a statistically significant difference in decapsulation time. We have performed the same test again focussing only on the `seedexpander` function and achieve very similar results.

We implemented the optimized attack against hqc-128 using an idealized timing oracle that reveals the number of `seedexpander` calls during the decapsulation. The attack may be implemented analogously for the other parameter sets. We set $N = 5$ for the number of samples from which a majority must be formed for each bit. We performed the attack 6096 times in 114 CPU core hours on a Ryzen 5900X with 64 GiB DDR4 3600 MT/s CL18 RAM. Each attack required a median of $866{,}143$ idealized timing oracle calls. Of the 6096 attacks 5315 were successful, yielding a success rate of more than 87 %. Among the failed attacks, approx. 26 % terminated with less than 3 incorrect bits in the secret key component $\mathbf{y}$. An additional brute-force step comprised of approx. $\sum_{i=0}^{3} \binom{17{,}669}{i} \approx 2^{40}$ offline decapsulations could therefore further boost the success probability. Furthermore, approx. 86 % of the failed attacks terminated with less than 20 incorrect bits and could therefore drastically reduce the security level of HQC. Thus, even if we are not able to recover all bits of the secret key we deem it likely that one can apply the known attacks to the HQC scheme which are listed in [Agu+21] as it will become feasible to solve the syndrome decoding problem or to mount structural attacks. We empirically determined the probability distribution of the number of incorrect bits after an attack and show the cumulative distribution function in Fig. 4.

4.2 Empirical Evaluation of the Attack on BIKE

The BIKE specification changed in some major ways between round 2 and 3 and there is now only a single BIKE variant with different security levels. The submitted version of the specification is 4.1. The specification has been updated during round 3 to version 4.2 in ways relevant to our attack; specifically the PRNG function used by the H function has been replaced. Though, the side-channel and

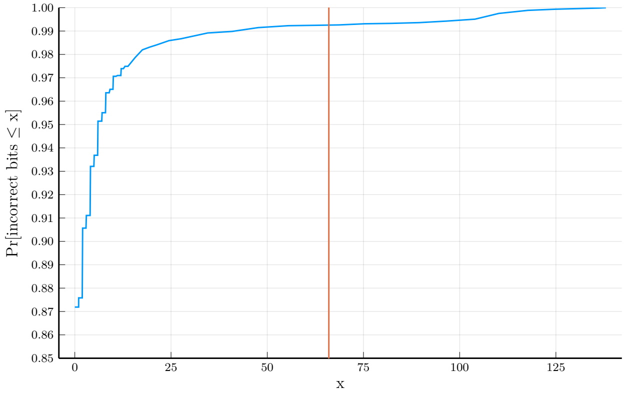


Figure 4: Empirical cumulative distribution function of the number of incorrect bits during the attacks. Approx. 87 % succeeded immediately. For those that failed additional post-processing steps could further improve the success probability. The vertical line indicates the weight of the secret key **y**. Less than 1 % of cases the attack terminated with more incorrect bits than bits are set in the secret key.

the presented attack remains, therefore the changes are discussed only briefly in section 4.4.

The presented version of the BIKE attack assumes the following pre-conditions:

1. It is possible to generate a decoding failure (and then use the *Error Amplification* attack to generate a chain of related decoding failures). This is possible due to

   - increasing the error weight when crafting the modified ciphertext artificially increases the DFR.
   - the lack of mandated weight-check on the error vector in the decapsulation stage of the BIKE specification.[6]

2. The timing profile of the H function depends on its input (value of m), i.e. it is not (or insufficently) protected against side-channels.

3. The attack requires a IND-CCA setting with static key re-use.

We now list some existing implementations and discuss the applicability of our attack:

[6] a weight check is discussed in the *Design Rational* chapter, but left out in the *Specification* chapter. It is mentionend in the IND-CCA security reduction to be implicit in the $\mathbf{e}' = \mathsf{H}(m')$ check, but not in the specification of H.

- **Reference implementation**: All versions of the reference implementation of BIKE fulfills these pre-conditions. But since it is not designed to protect against side-channels the existence of an attack is not unexpected.

- **Protected additional implementation**: It should be noted that there is no submitted official *additional implementation* in the submission package[7] for NIST PQC Round 3 version of the BIKE specification. The *additional implementation* folder in the Round 3 submission package is the protected implementation of the BIKE round 2 specification, version 3.2[8]. This version is vulnerable to the attack presented in this paper, with some minor modifications.

- **Github version**: Located at github.com/awslabs/bike-kem/ is another implementation, which has an additional weight check on the error vector (not given in the BIKE specification), located before the call to the H function. The end result is that in case of decoding failure *or* an error pattern of weight $\neq t$, the input to H is randomized. This renders the described attack much more difficult to exploit, since we are required to find a decoding failure without changing the weight of the error vector. On the other hand, the extra weight check opens up an even more efficient message recovery attack and we provide a very brief description of this in Section 4.3.

  Liboqs from the Open Quantum Safe Project [SM16] appears to use the same version of the BIKE implementation as the one above. Also, a recent **3rd party Intel Haswell** implementation due to [CCK21] that targets the Intel Haswell family of CPUs to achieve greater speeds than the official implementation appears to be based on the Github version and have copied the additional weight check.

- **3rd party ARM Cortex M4**: In the same paper [CCK21], the authors present a side-channel protected implementation targeting the ARM Cortex M4 processor. This version does not employ the additional weight check and is thus vulnerable to our attack.

The simulations and experiments related to BIKE in this paper is using the liboqs implementation for BIKE version 4.1, with the additional weight checks turned off. This enables us to verify our attack in a close to real world scenario. Ideally, the experiments would also be performed on the unmodified 3rd party ARM Cortex M4 implementation. Due to time-constraints however, we restricted ourselves to the Intel x86 platform-based implementations of the submitted BIKE version.

[7] obtained on 2021-12-31 from `https://csrc.nist.gov/CSRC/media/Projects/post-quantum-cryptography/documents/round-3/submissions/BIKE-Round3.zip`

[8] the same is true for the additional implementation found on the `bikesuite.org` website (Last checked 2021-12-31)

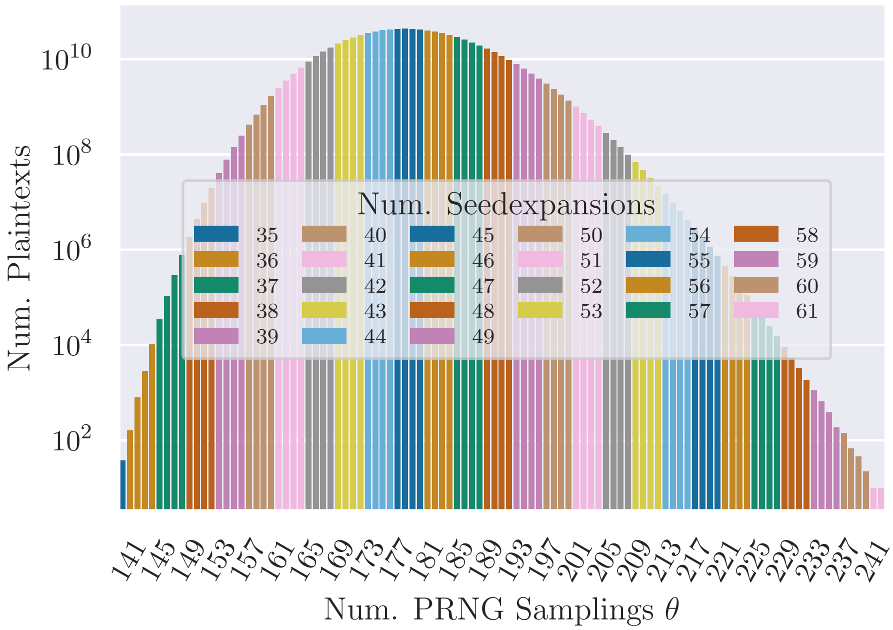


Figure 5: *BIKE-L1*: Distribution of the number of samplings θ in H to the underlying PRNG function as empirically simulated with approximately 10^{11} randomly generated plaintexts. Also presented are the number of seed expansions to the PRNG function.

The empirical investigation into *BIKE-L1* shows the number of expected rejections by the rejection sampling algorithm in Fig. 5. From the experiment we draw the following conclusions about the targeted implementation:

1. The number of PRNG samplings θ are equal to the number of sampled bit positions in $\mathbf{e_0}, \mathbf{e_1}$, therefore $\theta \geq T$.

2. θ, for *BIKE-L1*, has an expected value $E(\theta) \approx 178.6$, over the space of $\mathcal{M}$.

3. The number of rejections for *BIKE-L1* has an expected value of $E(\theta - T) = 44.6$

4. The experiment shows a skewed[9] normal distribution with a standard deviation of $\sigma = 7.714$

5. The underlying PRNG can serve 4 random samplings before requiring a new seed expansion.

[9] Skewed due to the influence of condition $\theta \geq T$

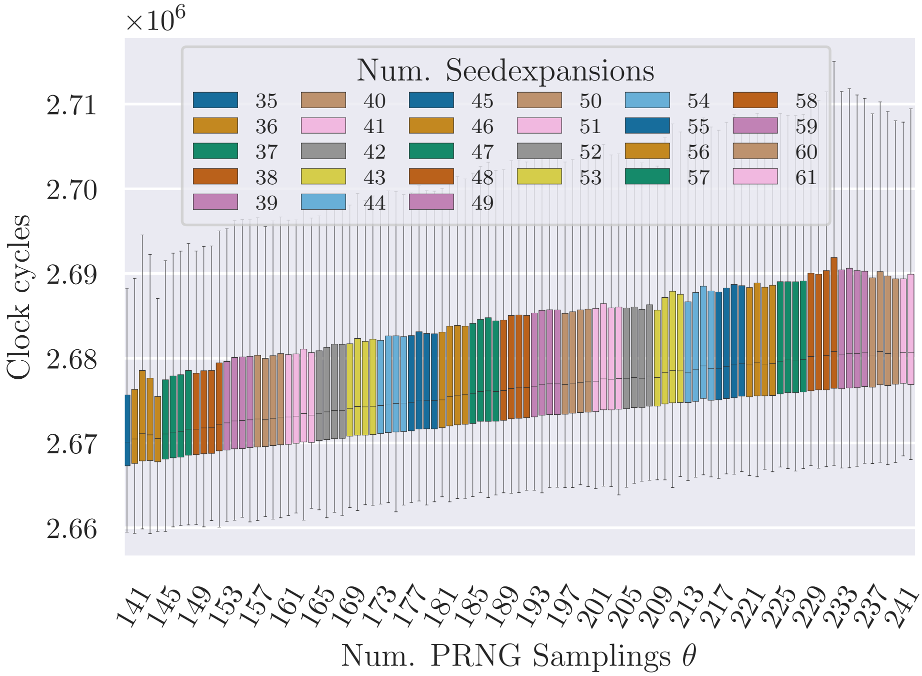


Figure 6: Box plots of 10^6 timing measurements of *BIKE-L1 Decaps* per value of θ. Values of θ with too few available plaintexts (< 10) are not simulated.

To determine the existence of the timing side-channel we observe the timing distributions of the decapsulation method (Algorithm 8) as a function of θ. As we can see in Fig. 6 it is indeed possible to distinguish between a high and low θ, although the variations are slight and we therefore require a relatively large number measurements for the distinguisher to give accurate outputs.

We performed the BIKE experiments in an HP EliteBook 820-G4 notebook with Intel Core i5-7200@2.50GHz and 8Gb RAM running on Ubuntu 20.04 LTS. We set Linux scaling governor to 'performance', turned off hyper threading, and turned off all extraneous processes.

Interestingly, the number of seed expansion calls does not appear to provide any noticeable influence on the runtime of the implemented rejection sampling algorithm. Consequently we must select a plaintext based on θ alone. Each seed expansion call is a simple call into AES which generally is implemented using the Advanced Encryption Standard New Instructions (AES-NI) CPU instruction set extension, which is very fast on modern CPUs.

As previously noted, since it is a one-time pre-computational cost, we can spend an almost arbitrary amount of computation looking for a good candidate plaintext which will provide us with a distinct timing profile in the rejection sampling algorithm.

Since we do not control what new plaintext is generated by decoding failures our candidate plaintext must cause a timing profile which is measurably distinct from other *likely* plaintexts. Likely, in this context, relates to the notion of the probability of making erroneous decisions for $m' \stackrel{?}{=} m$. That is, if $m' \neq m$ and $|\theta_m - \theta_{m'}| < \Delta\theta$, for some value $\Delta\theta$ for which the distinguisher is no longer reliable, then the distinguisher may output the wrong decision. The probability of the decoder randomly returning such an m' is the most important property to consider when determining the design and parameters of the distinguisher.

For the attack to succeed the probability of $|\theta_m - \theta_{m'}| < \Delta\theta$ must be minimized. This can be accomplished in two ways. First, by increasing the work-load of the pre-computation phase we can find a candidate value of m with θ_m as high/low as possible. The second way is to reduce the granularity ($\Delta\theta$) of the distinguisher by increasing the number of decapsulation measurements.

The probability of a distinguisher failure ϵ can easily determined by simulation. There are many ways to construct a distinguisher with a reasonably low failure rate ϵ. We selected a simple strategy where we use the minimum θ_m and where we use the 1% lowest measurement as the representative value for each distinguisher decision. This is done in order to select the value which is as close to noise free as possible.[10]

The distinguisher uses 2 phases; the profiling phase and the decision phase. In the profiling phase we first select (m, θ_m) and $\Delta\theta$ such that the probability of $|\theta_m - \theta_{m'}| >= \Delta\theta$, for a random $\theta_{m'}$, is less than the targeted ϵ. Then a large number of measurements are collected for plaintext m^* where $\theta_{m^*} = \theta_m + \Delta\theta$. The 1% lowest measurement is selected as a threshold.

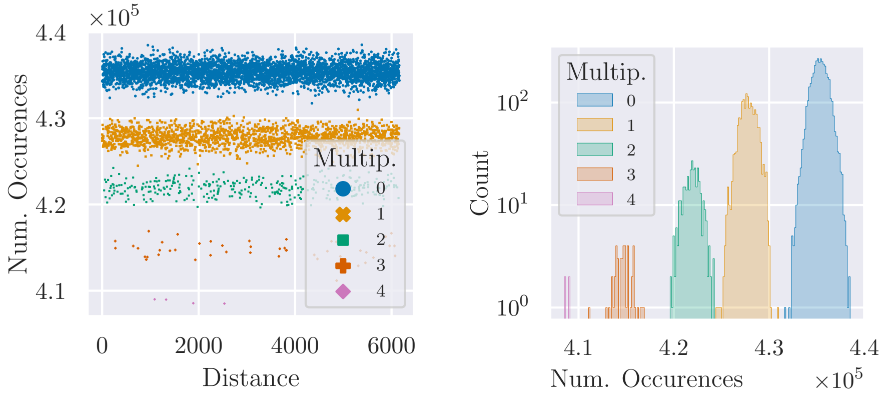
In the decision phase a number of decapsulation measurements are collected and, again, the 1% lowest value is selected. The selected value is compared against the previously selected threshold. If the measurement value is above the threshold we guess a decoding failure. If below, we determine decoding successful.

Clearly, the distinguisher can be made more sophisticated using statistical hypothesis testing or machine learning. However, for simply validating the practicality of exploiting this side-channel, this distinguisher is quite sufficient.

Simulations show that we can obtain $\epsilon \approx 0.01$ by constructing a distinguisher with $\Delta\theta = 22$. This value is obtained with a candidate plaintext m with $\theta_m = 138$ and $N_d = 1000$ decapsulation measurements, per decision. This was determined using the above parameters against 10^4 random error patterns of hamming weight 157, resulting in a DFR of 0.1369.

To complete the empirical evaluation of the attack we finally perform a full simulation of Algorithm 14. We have implemented the attack using an idealized

[10] We don't select the absolute minimum value as we have discovered that sometimes those values are impossible outliers. An hypothesis is that they come from instruction-reordering by the CPU and/or scheduling between CPU cores.



(a) Simulated aggregated distance spectrum using the *Error Amplification* attack. Listed in the legend is the various multiplicities $\mu(d)$ of the secret key.

(b) Plotting as histogram, the multiplicities $\mu(d)$ are quite separated and there should be no errors while determining the distance spectrum of the secret key.

Figure 7: Data is generated using $N_f = 8.5 \times 10^6$ decoding failures and a distinguisher with $\epsilon = 0.01$. For each distance is listed the number of occurrences that the specific distance was included in an error pattern that resulted in a decoding failure. The stratification into separate layers for each multiplicity is clearly visible.

oracle that output θ, the number of PRNG samplings performed by H. To show the real-world applicability of the attack the idealized oracle additionally simulates $\epsilon = 0.01$, artificially. The simulation results in the graphical representation of the distance spectrum of the secret key, as seen in Fig. 7.

Due to ϵ, a confirmation step was added where each found decoding failure was confirmed by an additional set of measurements. Otherwise the *Error Amplification* attack is sensitive to bad distinguisher decisions. Due to the chain of error patterns that is constructed it is critical that consecutive decoding failures are not missclassified. The extra confirmation step prevents this.

Observed in the figure is a clear picture of the distance spectrum without classification errors. This figure was obtained after about $N_f = 8.5 \times 10^6$ decoding failures in the decapsulation method for BIKE-L1. The simulation used a hamming weight of 149, which using the *Error Amplification* attack resulted in a DFR ≈ 0.146, a good match for our distinguisher above.

The final step is to do the key recovery, as detailed in [GJS16]. As in [GJS16], the reconstruction cost is negligible compared with the cost of querying the decryption oracles if the distance spectrum is fully recovered. The reason is that after quite few steps, the wrong guesses will be rejected with high probability and only the correct guess path will continue. One could balance the costs of building the distance spectrum and recovering the secret key, as done in [GJW19], by allowing errors in the recovered distance spectrum. As our aim is to demonstrate the new attack method, for simplicity, we leave this optimization trick for future works.

Finally, we estimate that to perform an actual key recovery we require about 6.7×10^{10} decapsulation measurements, or equivalently, 5.8×10^{7} number of idealized oracle calls. These numbers are given by:

$$\underbrace{\frac{N_f}{\mathrm{DFR}}}_{\substack{\text{Ideal}\\ \text{oracle}}} \times N_d + \underbrace{N_f \times N_d}_{\text{Confirmation}} = N_d \times N_f \times \frac{1+\mathrm{DFR}}{\mathrm{DFR}}.$$

We firmly believe that these numbers can be reduced, e.g., by an improvement of the distinguisher by statistical hypothesis testing, machine learning, or discarding ambiguous results, etc. Further options are optimizing the hamming weight (and thus the DFR) of the error patterns, allowing for larger ϵ and adding more confirmation steps, if necessary, allowing errors in the distance spectrum by trading for increased computational cost in the postprocessing stage, or spending further computational resources towards finding a more distinct (m, θ_m).

4.3 Message-Recovery from the New Weight Check

The described key-recovery timing attack on BIKE does not work for the new Github implementation and other related implementations (say in liboqs), due to an additional weight check on the error vector before the call to the H function, a check that is not explicitly specified. However, combined with the timing variation from the rejection sampling, this new weight check opens a new path for very efficient message recovery. In this section, we briefly describe such a message recovery attack.

We first build a distinguisher to distinguish if the weight check fails. In the new implementation, in Algorithm 8 after the call $\mathbf{e}' = \mathsf{Decode}()$ in line 1, the weight of $\mathbf{e}'$ is checked. If the weight is not t, $\mathbf{e}'$ is assigned a random value, otherwise it is kept. In line 2, $m' = c_1 \oplus \mathsf{L}(\mathbf{e}')$ and in the next line there is the call to $\mathsf{H}(m')$.

When we submit the same ciphertext to the BIKE decaps oracle multiple times, the input to the H function call are different random vectors if the weight check fails; otherwise, if the weight of $\mathbf{e}'$ is t, the inputs will be the same fixed vector. We can then build a distinguisher assuming that such execution time difference could be detected statistically after repeating the decaps oracle calls with the same ciphertext for many times.

With this distinguisher, we could launch a simple message-recovery attack. Assume that a correctly generated ciphertext $c = (\mathbf{c}_0, c_1) = (\mathbf{e}_0 + \mathbf{e}_1\mathbf{h}, m \oplus \mathsf{L}(\mathbf{e}_0, \mathbf{e}_1))$ is received. One can now flip the first and the i^{th} bits of $\mathbf{c}_0$ (i.e., flipping the first and the i^{th} bits of unknown $\mathbf{e}_0$) and send the new ciphertext to the decaps oracle multiple times. Even though the decaps output is meaningless, from the distinguisher using the timing measurement one can detect if the flipped two bits have the same value. If they have the same value, the weight of $\mathbf{e}'$ will increase by 2, but if they have different values the weight of $\mathbf{e}'$ will be t. Since $\mathbf{e}_0$ is an extremely sparse vector, if we have more pairs among the $(r-1)$ pairs to be the same, the first bit of $\mathbf{e}_0$ should be 0; otherwise, it should be 1. When the first bit of $\mathbf{e}_0$ is decided, one can estimate the i^{th} bit of $\mathbf{e}_0$.

Note that one could also flip the first bit of $\mathbf{c}_0$ and add the vector $\mathbf{E}_\mathbf{i}\mathbf{h}$ to $\mathbf{c}_0$, where $\mathbf{E}_\mathbf{i}$ is the vector with only the i^{th} position nonzero. This is equivalent to flip the first bit of $\mathbf{e}_0$ and the i^{th} bit of $\mathbf{e}_1$. Thus, the value of $\mathbf{e}_1$ can be roughly estimated. We can then employ a post-processing step with ISD (information set decoding) algorithms to correct some distinguishing errors in the previous steps.

The attack was not implemented since the additional check is not part of the specification, but we can do a rough estimate of the complexity of such an attack. If we do a few hundred oracle calls with the same ciphertext and run through $2r$ different modified ciphertexts it appears very likely that the attack would be successful, since it would allow for some distinguishing errors that an ISD approach would then correct. In total, we may use several million oracle calls, which is much less than the described key-recovery attacks. We leave the investigation of its exact performance for future work. The conclusion is that the added weight check as it is done in the implementation is only weakening the security of the scheme.

4.4 Updates to the BIKE Specification, Version 4.2

As already mentioned, during Round 3 of the NIST PQC standardization effort, the BIKE specification has been updated from the submitted version 4.1 to the latest version 4.2. The specific change that is relevant to our attack is the change from WAES-CTR-PRF to WSHAKE256-PRF as the pseudo random bit generation algorithm. Also hash functions K and L now use SHA3-384 instead of the older SHA384.

The change was done to unify the dependencies of the symmetric primitives, now the only dependency is the underlying KECCAK sponge construction, whereas previously both the AES-CTR and SHA-384 primitives were required.

The change is beneficial in the eye of an attacker, at least in software and in the short term, because the lack of hardware acceleration for the KECCAK primitive in contemporary CPUs makes the new PRF slower. As a slower PRF is easier to profile, the number of measurements required should, theoretically, be reduced proportionally. We leave the confirmation of our hypothesis for future work.

5 Discussion on Countermeasures

To counter our proposed attacks and to remove the exploitable leakage, we see two ways. When analysing the construction of the KEM version of HQC, one might argue that finding a different tranformation from the IND-CPA PKE that eliminates the re-encryption step in the decapsulation might solve the problem.

BIKE actually manages to optimize away the re-encryption step using the HHK implicit rejection transform, but still needs to retain the call to the constant-weight hashing funtion to check for decryption errors. Exploring different options for IND-CCA transformations in both schemes might be interesting future work. Short of finding a different IND-CCA transformation, the sampling of a fixed weight bit vector must be implemented isochronously.

We focus on a constant-time implementation of the algorithm. Since both attacks we presented exploit a structurally similar side-channel the countermeasure we present could be applied to both HQC and BIKE. We implemented and evaluated the countermeasure for HQC only. We identify two avenues for implementing a low fixed-weight vector sampling algorithm that is constant-time in the used seed. For the first one we initialize the vector of length n starting with a run of w set bits. Then we shuffle the array. This will result in a random vector of weight w. To shuffle the array one could use, e.g., the Fisher-Yates shuffling algorithm as described in [Knu97, p.145]. A naïve implementation of Fisher-Yates shuffling will leak timing information, as it will use secret-dependent array accesses. Using generic methods to make these array accesses constant-time results in an unacceptable asymptotic time complexity. Nicolas Sendrier presents a sophisticated approach how the Fisher-Yates shuffle can be modified to reduce the time complexity specifically for the use-case of generating random low fixed-weight vectors [Sen21]. For the case of BIKE, he shows that a small bias in the sampling distribution has a negligible impact on the security of the scheme. His work was published as a response to the initial pre-print version of our work, which only contained the attack on HQC and its countermeasure. While we are not aware of an implementation and concrete evaluation, the decreased time complexity of Sendrier's approach makes the method very compelling. Our approach requires little deviation from the already existing code-base, and allows existing implementations to be patched with relative ease. Another approach would be to use a reverse sorting algorithm, using an established sorting algorithm like merge-sort, as it is proposed in [WSN18] for a Classic McEliece hardware implementation. The reverse merge-sort induces a slight bias which is solved by a rejection and is therefore not suitable for a constant-time implementation, unless one can show that the bias is acceptable. The Benes☒network used in the C reference implementation of Classic McEliece is aligned to a vector size of a power of 2 which is not the case in HQC.

We propose an approach that samples the specified number of distinct bit positions in constant-time and sets the bits in the vector in constant-time. For HQC, only the distinct position sampling is not constant-time. Our modification results in an algorithm that is only probabilistically correct and may sample too few distinct bit positions. The probability of this failure mode can be chosen arbitrarily small and made negligible.

To obtain our final countermeasure we perform several modifications to the algorithm. After each modification the algorithm is a fully functioning algo-

```
 void vect_set_random_fixed_weight(
   seedexpander_state *ctx,
   __m256i *v256, uint16_t weight) {
-  size_t random_bytes_size = 3 * weight;
+  size_t random_bytes_size = 2 * 3 * weight;
-  uint8_t rand_bytes[3 * PARAM_OMEGA_R] = {0};
+  uint8_t rand_bytes[2 * 3 * PARAM_OMEGA_R] = {0};
```

Figure 8: Patch to HQC to remove additional seedexpander calls

rithm, however has different side-channel behavior. The first modification in Section 5.1 is very simple: we adjust the size of the randomness buffer to prevent the seedexpander function from being called a varying number of times in a single Sample call. It works by finding a loose upper bound for the number of bytes that the entire Sample function requires. Following, in Section 5.2 we replace a rejection sampling algorithm used to generate random positions in the range $\{0, \ldots, n-1\}$ with a constant time algorithm using modulo reduction of a large number. In Section 5.3 we then detail how the loose upper bound for the number of bytes to sample used in Section 5.1 can be tightened by accurately modeling the distribution of the number of times a number is sampled. We compute parameters required to make the probability that the number of bytes sampled is insufficient negligible. The resulting parameters depend upon whether the countermeasure in Section 5.2 was applied (κ_1) or not (κ_{p_s}), as the original version can fail to sample a number (in the case of a rejection) and our countermeasure always succeeds. As a final modification in Section 5.4 we modify the the loop that generates random distinct positions to always perform the same number of iterations κ resulting in a constant-time algorithm that can fail to produce a result of the expected weight with negligible probability. The algorithm is approximately correct, because we always perform κ iterations, even if we would need more iterations because e.g. many bit positions collided with previously sampled ones. Lastly, we benchmark the resulting countermeasure and compare it to the original in Section 5.5.

5.1 Remove Additional seedexpander Calls

The first attempt we make to get a countermeasure is to eradicate the concrete side-channel that we use for the attack. The rejection sampling algorithm generates new random data using the seedexpander function on demand. It is vanishingly unlikely that a single Sample invocation induces more than one additional seedexpander call. Therefore, our first, obvious countermeasure is to increase the number of bytes that are generated initially to double the previous amount. This results in a patch to the sampling function shown in Fig. 8. However, the algorithm is not constant-time: rejection sampling still performs a different number of iterations depending on the message and each random number is also sampled using rejection-sampling. While the countermeasure increases the effort required to perform the attack, it could still permit attacks in a low-noise environment to recover the key. Further, for BIKE the number of seedexpansions is not as large a factor for the timing distribution and therefore such a patch would not have a lot of impact for BIKE.

5.2 On Constant-Time Random Number Generation

To further reduce the timing leakage we can try to remove the inner rejection used for generating random indices into the vector. The inner rejection sampling is detailed in Algorithm 16. Instead of rejection sampling integers in the range $0 \leq x < \lceil 2^k/m \rceil m$, we generate $b \gg \log_2 m$ random bits and then reduce the generated integer modulo m to the desired range. This will bias the resulting distribution if m does not divide 2^b, which is the case here. Therefore, we need to pick a sufficiently large b for the statistical distance to be negligible. In particular we are interested in minimizing the statistical distance (SD) between the uniform distribution over $\{0, \ldots, m-1\}$ and the distribution generated by $x \bmod m$ where x is drawn uniformly at random from $\{0, \ldots, 2^b - 1\}$. We define the statistical distance between two probability distributions X and Y over some discrete domain Ω to be:

Algorithm 16 Inner Rejection Sampling Algorithm

```
Output: Random number in $[0, \ldots, m-1]$
repeat
    $i \xleftarrow{\$} [0, 2^k)$
until $i < \left\lceil \frac{2^k}{m} \right\rceil m$
return $i \bmod m$
```

$$\mathrm{SD}_{X,Y} = \frac{1}{2} \cdot \sum_{z \in \Omega} |\Pr[X = z] - \Pr[Y = z]|$$

Let U_m be the uniform probability distribution over $\{0, \ldots, m-1\}$:

$$\Pr[U_m = z] = \begin{cases} \frac{1}{z} & \text{if } 0 \leq z < m \\ 0 & \text{otherwise} \end{cases}$$

Table 5: Statistical distance between the uniform distribution over $\{0,\ldots,m-1\}$ and the distribution of random integers from 0 to $2^b - 1$ reduced modulo m for hqc-128 ($n = 17{,}669$).

b	$\log_2 \mathrm{SD}_{U_m,M_{2^b}}$ (approx.)
16	-4
32	-20
64	-52
128	-116
256	-244
512	-502

Additionally, we define the probability distribution M_n which reduces an integer in $\{0,\ldots,n-1\}$ modulo m. Its probability distribution is given by:

$$\Pr[M_n = z] = \begin{cases} \frac{\lceil n/m \rceil + 1}{n} & \text{if } 0 \leq z < n \bmod m \\ \frac{\lceil n/m \rceil}{n} & \text{if } n \bmod m \leq z < m \\ 0 & \text{otherwise} \end{cases}$$

The statistical distance between these two distributions is:

$$\mathrm{SD}_{U_m,M_n} = \frac{1}{2} \cdot \sum_{z \in \{0,\ldots,m-1\}} \left|\Pr[U_m = z] - \Pr[M_b = z]\right| \tag{7}$$

$$= \frac{1}{2} \cdot \Bigg((n \bmod m) \cdot \left| \frac{1}{m} - \frac{\left\lceil \frac{n}{m} \right\rceil + 1}{n} \right| + \tag{8}$$

$$(m - (n \bmod m)) \cdot \left| \frac{1}{m} - \frac{\left\lceil \frac{n}{m} \right\rceil}{n} \right| \Bigg)$$

In Table 5 we computed the statistical distance between the uniform distribution and the modular reduction technique for various numbers of bits b. The parameter m is the length of the vector in HQC. We leave the choice of an acceptable statistical distance to the designers of the scheme. For our further testing we use $b = 128$.

We can implement a modular reduction of a b-bit non-negative number x modulo a small number efficiently using basic rules of modular arithmetic. We can represent x in e.g. base 2^8 as $x = x_0 + 2^8 \cdot x_1 + 2^{8\cdot 2} \cdot x_2 + \cdots + 2^{8\cdot(\ell-1)} x_{\ell-1} + 2^{8\cdot\ell} \cdot x_\ell$ where $\ell = \left\lceil \frac{b}{8} \right\rceil$. We split up the computation of $x \bmod m$ in the following way:

$$x \bmod m = \left(\cdots \overbrace{\left(x_{\ell-1} + 2^8 \cdot \underbrace{(x_\ell \bmod m)}_{z_0} \right)}^{z_1} \bmod m \cdots \right) \bmod m$$

Generalizing this, we can write an iterative algorithm that computes in iteration i:

$$z_i = \begin{cases} x_\ell \bmod m & \text{if } i = 0 \\ (x_{\ell-i} + 2^8 \cdot z_{i-1}) \bmod m & \text{otherwise} \end{cases}$$

and $z_\ell = x \bmod m$. We can implement this algorithm for a random number x where each x_i is drawn from `rand_bytes` as shown in Listing 1.

Listing 1 Constant time modulo reduction of $x \mod m$ in multiple steps where `BYTES_PER_INDEX` is $\lceil \frac{b}{8} \rceil$.

```
uint32_t random_data = 0;
for (uint32_t k = 0; k < BYTES_PER_INDEX; ++k) {
  random_data = ((uint32_t)rand_bytes[j++] | (random_data << 8));
  random_data %= PARAM_N;
}
```

Additionally, while a divide instruction is not constant-time in general on most Instruction Set Architectures (ISAs), reducing modulo a constant is optimized by the compiler into a sequence of instructions that can be executed in constant time. The optimization performed by the compiler is a Barrett reduction [MOV97, p.603]. This can be observed in Listing 2. Here the compiler replaced the `idiv` instruction by a series of shifts, additions and multiplications. All of these instructions complete with a fixed latency on the Zen 3 ISA according to Agner's instruction tables[11]. To ensure that the compiled result always uses these instructions, which we have verified to be constant-time, we can copy the compilation result into an `__asm__ volatile` block.

5.3 Tight Upper Bound on the Number of Samples Required

We wish to minimize the number of random bytes generated, while still ensuring that we never run out of randomness during the rejection sampling function so that we only have to perform seedexpansion once at the beginning of the function. To this end, we analyze the probability of requiring a certain number of iterations

[11] `https://www.agner.org/optimize/instruction_tables.pdf`, accessed on 2021-11-05.

Listing 2 Modular reduction of an integer a modulo a constant in C and the resulting Intel-style x86 assembly with optimization level 2 using clang.

```c
#include <stdint.h>

uint32_t f(uint32_t a) {
    return a % 23869;
}
```

```asm
f:
    mov eax, edi
    mov ecx, edi
    mov edx, 2948122845
    imul    rdx, rcx
    shr rdx, 46
    imul    ecx, edx, 23869
    sub eax, ecx
    ret
```

in the rejection sampling algorithm. We introduce the random variable X_{n,i,p_s}, which is the number of distinct elements after attempting to sample i elements from $\{1,\ldots,n\}$ with each sample succeeding with the probability p_s. The success probability p_s can be used to model the case where the inner rejection sampling algorithm has to retry sampling an element from $\{1,\ldots,n\}$, because the sampled element was not in the required range. Therefore, if a sample fails, it increases the number of iterations, but no element is sampled. This yields the following recursive relation:

$$\Pr[X_{n,i,p_s} = w] = \begin{cases} 0 & \text{if } i < w \\ 1 & \text{if } w = i = 0 \\ p_s & \text{if } w = i = 1 \\ Eq.\ (10) & \text{otherwise} \end{cases} \tag{9}$$

$$p_s \frac{w}{n} \Pr[X_{n,i-1,p_s} = w] + (1 - p_s \max(0, \frac{w-1}{n})) \Pr[X_{n,i-1,p_s} = w-1] \tag{10}$$

We are now sufficiently equipped to compute the probability that the rejection sampling algorithm requires $\leq i$ iterations to sample w distinct bit positions. This query is equivalent to the probability, that after i iterations $\geq w$ distinct bit positions have been sampled. We can compute this by simply summing over the number of distinct positions:

$$\Pr[X_{n,i,p_s} \geq w] = \sum_{x=w}^{i} \Pr[X_{n,i,p_s} = x]$$

Finally, we define the random variable U_{n,w,p_s} to be the number of iterations required to sample w distinct elements out of $\{1,\ldots,n\}$ with each sample suc-

Table 6: Number of indices κ that must be derivable from the generated randomness reservoir to achieve a probability on the order of the security parameter of a message emitting multiple seedexpander calls. Here, p_s is set to $\lceil 2^k/m \rceil m/2^k$ for when the original rejection sampling is used or 1 when the bit position sampling cannot fail due to the use of the constant-time random number generation scheme.

Instance	κ_{p_s}	$\log_2(1-(\Pr[U_{n,\omega_r,p_s} \leq \kappa_{p_s}])^3)$	κ_1	$\log_2(1-(\Pr[U_{n,\omega_r,1} \leq \kappa_1])^3)$
hqc-128	99	≈ -134	97	≈ -129
hqc-192	152	≈ -193	146	≈ -195
hqc-256	192	≈ -261	190	≈ -259

ceeding with probability p_s. Then, the probability of requiring $\leq i$ iterations is:

$$\Pr[U_{n,w,p_s} \leq i] = \Pr[X_{n,i,p_s} \geq w]$$

We can use the random variable U_{n,w,p_s} to minimize the number of random bytes that we need to sample. The probability that a message emits ≥ 1 additional seedexpander calls when the randomness reservoir provides sufficient entropy for κ random indices is:

$$1 - (\Pr[U_{n,\omega_r,p_s} \leq \kappa])^3.$$

We would like this probability to be negligible. We can compute a suitable κ for which the probability is $\leq 2^{-\lambda}$ where λ is the security parameter. This is done by increasing κ until the probability is low enough. The number of iterations depends on the success probability of sampling a random index. When we retain the original inner rejection sampling algorithm we use the success probability p_s to compute κ_{p_s}. For the constant-time random number generation we use a success probability of 1 to compute κ_1. Note that these probabilities are high enough for these messages to feasibly exist. However, we deem it infeasible to compute such messages, as they are so rare.

The results of these computations can be seen in Table 6. Using κ we can optimize the countermeasure to generate the least amount of randomness to eradicate additional seedexpander calls. Note that $\kappa_1 \leq \kappa_{p_s}$, since the rejection sampling algorithm requires less iterations when every random number generation succeeds. However, the constant-time Random Number Generator (RNG) still requires much more random bytes to be generated, since it requires 16 bytes per index, instead of approx. 3 in expectation.

We can further optimize the runtime of the RNG by using the full width of the ISA's registers. Instead of reducing one byte at a time we can reduce 4 bytes at once by using 64 bit registers and multiplying each intermediate result z_{i-1} by $2^{8 \cdot 4}$, as we detail in Listing 3. Further performance improvements may be achievable

Listing 3 Optimization in the random number generation by reducing 4 bytes at once.

```
uint32_t rand_bytes[BYTES_PER_INDEX * K_1 / 4] = {0};
// [...]
uint64_t random_data = 0;
for (uint32_t k = 0; k < BYTES_PER_INDEX / 4; ++k) {
  random_data = (uint64_t) rand_bytes[j++] + (random_data << 32);
  random_data %= PARAM_N;
}
```

through the use of even wider registers or SIMD instructions to produce multiple positions at once.

5.4 Constant-Time Monte-Carlo

We can now forge a constant-time algorithm that is approximately correct using minimal modifications. It fails to produce a correct result with an error-probability that we can choose to be arbitrarily low. The first step is to always produce the same number of random positions into the generated vector. Additionally, for each position we keep track of whether it is needed, i.e., whether the generated index has already been sampled before and whether we have already sampled enough unique indices. Using this information, we can then set the bit only if it is needed – in constant time. However, if we fail to sample enough unique indices, the algorithm may produce a vector of too low weight. We cannot catch this error and try again, as that would introduce a timing-variability. Therefore, we must sample enough positions such that this case does not happen with overwhelming probability. We can reuse the κ_1 listed in Table 6 for this purpose. Using these parameters the probability that we sample a vector of too low weight is $\leq 2^{-\lambda}$, where λ is the security parameter.

Concretely, we keep track of the number of unique positions sampled and whether we need each position by through the `count` variable and the `take` array. We then sample κ_1 positions from $\{0, \ldots, n-1\}$. Instead of trying to sample a position again when a position is not unique, we store it unconditionally but keep track of whether we need the position. The modifications are seen in Listing 4. The `exist` variable is 1 iff the position has not been sampled before and `i` is the iteration count in $\{0, \ldots, \kappa_1 - 1\}$. We refer to the vector of n bits that is modified by the algorithm as the *bit-array*.

The next phase of the algorithm uses Advanced Vector Extensions (AVX2) instructions to set the sampled bit positions in the bit-array. This algorithm is vectorized to process the bit-array in 256 bit chunks. We modify this algorithm to only include a position if `take[i]` is set by computing a bit mask that is 1^{256}

Listing 4 Modifications to the position sampling

```
uint32_t count = 0;
uint8_t take[K_1];
// [...]
tmp[i] = random_data;
uint8_t not_enough = count < weight;
uint8_t needed = (!exist) & not_enough;
take[i] = needed;
count += needed;
```

if `take[i] == 1` and 0^{256} otherwise. We then modify the first loop to compute the bitwise and of `bit256[i]` and the mask stored in `take256`. This results

Listing 5 Modifications to the set-bit placing algorithm

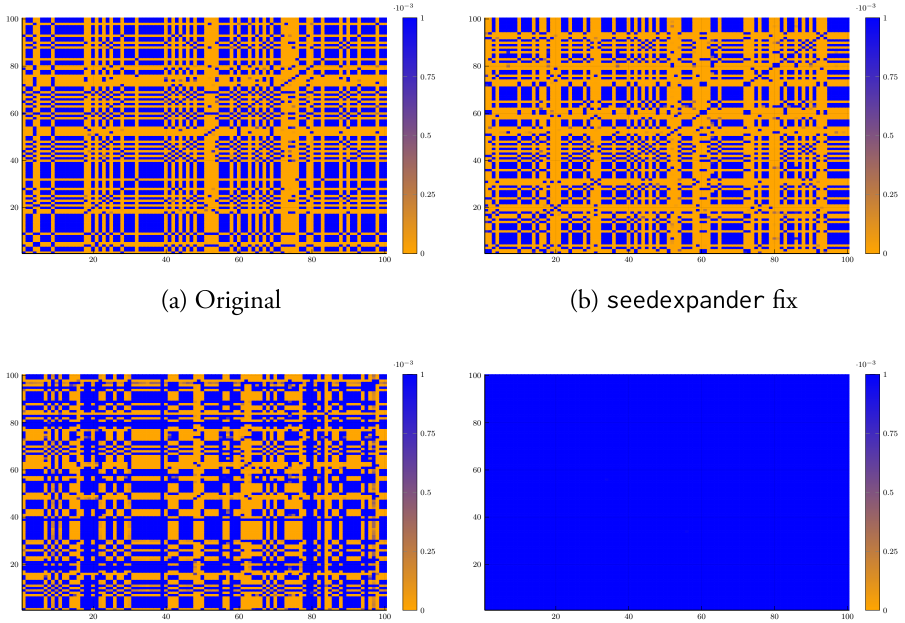
```
__m256i take256 = _mm256_set1_epi64x(take[i]) == 1;
bit256[i] = bloc256&mask256&take256;
```

in `bit256[i]` being 0^{256} if the bit is not needed. When this 256 bit vector is later xor-ed with the `aux` variable, it will have no impact, since $0 \oplus x = x$. The modifications are seen in Listing 5.

5.5 Evaluation of the Proposed Countermeasures

The side-channel evaluation results can be viewed in Fig. 9. The number of the detected difference clearly diminishes as more of the suggested modifications are applied. In particular, the final countermeasure eradicates all statistically significant timing differences in the `Sample` function as can be seen in Fig. 9d. We conclude that the final countermeasure eradicates all timing-leakage that we could detect from the algorithm with respect to the seed used by the XOF.

We measure the number of cycles the Sample function requires for random messages for the original and the three patched versions to evaluate the performance impact of the additional instructions. We obtained 1 million measurements and removed outliers that deviate more than 3 standard deviations from the mean. Additionally, we gave the process a niceness of -20 on a dedicated machine. The process is pinned to a single core, and all other processes are pinned to different cores. The results may be seen in Table 7. We collected the mean and median number of cycles. The median number of cycles is increasing with more patches applied. We can see that the RNG fix is extremely costly in terms of cycle count and together with the `seedexpander` fix induces a 22.8 % increase in the median number of cycles. The main fault is likely that the constant-time RNG method



(a) Original

(b) `seedexpander` fix

(c) `seedexpander` + RNG fix

(d) `seedexpander` + RNG fix + constant iters

Figure 9: P-values for Welch's t-test testing whether there is a statistically significant difference between the computation time of the part invoking the Sample function in the re-encryption of the decapsulation for each pair in 100 ciphertexts generated for a single keypair. Orange indicates that a statistically significant difference was detected. The final countermeasure eradicates all statistically significant timing differences in the Sample function.

generates and processes approximately 5 times the number of random bytes. Furthermore, we observe that the `seedexpander` patch alone is extremely cheap and only incurs a 1 % increase in the number of cycles.

While fixing the `seedexpander` side-channel is cheap, it is not sufficient to obtain constant-time code. We were able to use the constant-time RNG in the design of further algorithms. Unfortunately, the constant-time RNG comes with a heavy performance hit, and it is not trivial to decide on a number of bytes to consume for each generated position. The final modification is constant-time, however it has a non-zero probability of returning an incorrect result. We choose this probability low enough for this to likely not be a practical issue.

Version	Median Cycles
original	259,370 (+ 0.0%)
`seedexpander` fix	261,849 (+ 1.0%)
`seedexpander` + RNG fix	318,533 (+22.8%)
`seedexpander` + RNG fix + constant number of iterations	334,628 (+29.0%)

Table 7: Benchmark results in number of cycles for the modifications of the rejection sampling algorithm. Modifications tested on hqc-128. Cycle counts include the entirety of the decapsulation function.

6 Conclusions, Lessons, and Future Work

We have presented novel key-recovery timing attacks on HQC and BIKE, where non-constant-time rejection sampling procedures are implemented for generating random vectors with a specific weight. The time differences caused by rejection sampling could leak whether the input message to the deterministic re-encryption procedure (or to a hash function) in the IND-CCA transformation is unchanged. Such secret information is sufficient for recovering the secret key of HQC and BIKE.

The considered implementation of HQC in this work has been found vulnerable despite the claim of the authors of HQC that the recent code is thoroughly analyzed so that only unused randomness (i.e., rejected based on public criteria) or otherwise nonsensitive data is leaked. The identified vulnerability probably has been hidden from scrutiny because the modular design of the HQC KEM employing the FO transformation conceals the dependence of the secret key to the rejection sampling function, due to a subtle error in the specification. In the IND-CPA version of HQC, encryption is non-deterministic, and thus the variations of the employed rejection sampling function is of no concern. The KEM/DEM version of HQC, as specified in Figure 3 in the specification, invokes a slightly different HQC.PKE encryption scheme than the one described in Figure 2 of the specification: one that fixes the randomness to make encryption deterministic. Only because the re-encryption in the KEM decapsulation is deterministic and because the seed is derived from secret data, non-constant-time rejection sampling becomes a problem. This highlights the issue of providing high level definitions of a cryptosystem: the definition is good enough for an implementer to get the functionality correct, but hides from manual inspection the ominous dependence identified and exploited in this work. However, in the case of HQC the specification encourages the use of the exploited rejection sampling algorithm. Therefore, the flaw we identify would likely be implemented by any implementer. This problem also highlights the need for automated, possibly standardized tools to check implementations for secret-dependent timing variations.

Regarding BIKE, we have identified a timing variation very similar to the one discovered in HQC. This vulnerability can be exploited for a key-recovery attack on BIKE. We found several vulnerable implementations, including the implementations in the NIST round 3 submission. Interestingly, in the most recent versions from Github, an additional weight check before the re-encryption procedure is employed, which can make the current key-recovery attack version unpractical. We emphasize that this additional weight check actually weakens the security of BIKE, since such a weight check allows more efficient message-recovery attacks. We still suggest to have a fully constant-time implementation of BIKE.

Our proposed countermeasure does incur a heavy performance degradation. However, it does eliminate all timing-variations that we could detect from the analyzed function. The constant-time variant of the Fisher-Yates algorithm proposed by Sendrier in a parallel work to ours introduces a slight bias in the uniform distribution but without an impact on the security properties of the scheme. It is another very interesting approach and should be considered in upcoming implementations and research activities as well. Future work could focus on improving the performance of the mentioned countermeasures through the use of SIMD instructions or different algorithms.

Acknowledgements

We would like to thank Dr. Gustavo Banegas for his shepherding care and the reviewers for their valuable feedback. The work presented in this paper has been partly funded by the German Federal Ministry of Education and Research (BMBF) under the project "QuantumRISC" (ID 16KIS1033K) [Qua20], by the Swedish Research Council (Grants No. 2019-04166 and No. 2021-04602), by the Swedish Civil Contingencies Agency (Grants No. 2020-11632), by the Swedish Foundation for Strategic Research (Grant No. RIT17-0005), and by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations and simulations were partly enabled by resources provided by LUNARC.

References

[Agu+] C. Aguilar Melchor et al. *Optimized implementation of HQC.* available at:`https://pqc-hqc.org/download.php?file=hqc-optimized-implementation_2021-06-06.zip`.

[Agu+18] C. Aguilar-Melchor, O. Blazy, J. C. Deneuville, P. Gaborit, and G. Zemor. "Efficient Encryption from Random Quasi-Cyclic Codes". In: *IEEE Transactions on Information Theory* 64.5 (2018), pp. 3927–3943.

[Agu+21] C. Aguilar Melchor et al. *HQC*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. National Institute of Standards and Technology, 2021.

[AP13] N. J. AlFardan and K. G. Paterson. "Lucky Thirteen: Breaking the TLS and DTLS Record Protocols". In: *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*. IEEE Computer Society, 2013, pp. 526–540.

[Ape+15] G. I. Apecechea, M. S. Inci, T. Eisenbarth, and B. Sunar. "Lucky 13 Strikes Back". In: *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '15, Singapore, April 14-17, 2015*. Ed. by F. Bao, S. Miller, J. Zhou, and G. Ahn. ACM, 2015, pp. 85–96.

[Ara+21] N. Aragon et al. *BIKE*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. National Institute of Standards and Technology, 2021.

[Bae+19] C. Baetu, F. B. Durak, L. Huguenin-Dumittan, A. Talayhan, and S. Vaudenay. "Misuse Attacks on Post-quantum Cryptosystems". In: *Advances in Cryptology – EUROCRYPT 2019, Part II*. Ed. by Y. Ishai and V. Rijmen. Vol. 11477. Lecture Notes in Computer Science. Darmstadt, Germany: Springer, Heidelberg, Germany, May 2019, pp. 747–776.

[BB05] D. Brumley and D. Boneh. "Remote timing attacks are practical". In: *Computer Networks* 48.5 (2005), pp. 701–716.

[BDL01] D. Boneh, R. A. DeMillo, and R. J. Lipton. "On the Importance of Eliminating Errors in Cryptographic Computations". In: *Journal of Cryptology* 14.2 (2001), pp. 101–119.

[BDL97] D. Boneh, R. A. DeMillo, and R. J. Lipton. "On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract)". In: *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*. Ed. by W. Fumy. Vol. 1233. Lecture Notes in Computer Science. Springer, 1997, pp. 37–51.

[Ben17] S. Benjamin. *Perspectives on the State of Affairs for Scalable Fault-Tolerant Quantum Computers and Prospects for the Future*. Presented at the 5th ETSI-IQC Workshop on Quantum-Safe Cryptography. available at https://docbox.etsi.org/Workshop/2017/201709_ETSI_

`IQC_QUANTUMSAFE/TECHNICAL_TRACK/S03_THREATS/UNIofOXFORD_BENJAMIN.pdf`. 2017.

[BL16] D. J. Bernstein and T. Lange. "Failures in NIST's ECC standards". In: (2016). available at: `https://cr.yp.to/newelliptic/nistecc-20160106.pdf`, pp. 1–27.

[Bru+16] L. G. Bruinderink, A. Hülsing, T. Lange, and Y. Yarom. "Flush, Gauss, and Reload - A Cache Attack on the BLISS Lattice-Based Signature Scheme". In: *Cryptographic Hardware and Embedded Systems – CHES 2016*. Ed. by B. Gierlichs and A. Y. Poschmann. Vol. 9813. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2016, pp. 323–345.

[BT11] B. B. Brumley and N. Tuveri. "Remote Timing Attacks Are Still Practical". In: *Computer Security - ESORICS 2011 - 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings*. Ed. by V. Atluri and C. Díaz. Vol. 6879. Lecture Notes in Computer Science. Springer, 2011, pp. 355–371.

[CCK21] M.-S. Chen, T. Chou, and M. Krausz. "Optimizing BIKE for the Intel Haswell and ARM Cortex-M4". In: *IACR TCHES* 2021.3 (2021). `https://tches.iacr.org/index.php/TCHES/article/view/8969`, pp. 97–124.

[CWR09] S. A. Crosby, D. S. Wallach, and R. H. Riedi. "Opportunities and Limits of Remote Timing Attacks". In: *ACM Trans. Inf. Syst. Secur.* 12.3 (2009), 17:1–17:29.

[Dur+14] Z. Durumeric et al. "The Matter of Heartbleed". In: *Proceedings of the 2014 Internet Measurement Conference, IMC 2014, Vancouver, BC, Canada, November 5-7, 2014*. Ed. by C. Williamson, A. Akella, and N. Taft. ACM, 2014, pp. 475–488.

[Dya18] M. Dyakonov. *The Case Against Quantum Computing*. IEEE Spectrum. available at: `https://spectrum.ieee.org/the-case-against-quantum-computing`. 2018.

[Gab21] P. Gaborit. *Personal Communication*. Nov. 2021.

[Gal62] R. G. Gallager. "Low-density parity-check codes". In: *IRE Trans. Inf. Theory* 8.1 (1962), pp. 21–28.

[GJN20] Q. Guo, T. Johansson, and A. Nilsson. "A Key-Recovery Timing Attack on Post-quantum Primitives Using the Fujisaki-Okamoto Transformation and Its Application on FrodoKEM". In: *CRYPTO 2020, Part II*. Ed. by D. Micciancio and T. Ristenpart. Vol. 12171. LNCS. Springer, Heidelberg, Aug. 2020, pp. 359–386.

[GJS16] Q. Guo, T. Johansson, and P. Stankovski. "A Key Recovery Attack on MDPC with CCA Security Using Decoding Errors". In: *Advances in Cryptology – ASIACRYPT 2016, Part I*. Ed. by J. H. Cheon and T. Takagi. Vol. 10031. Lecture Notes in Computer Science. Hanoi, Vietnam: Springer, Heidelberg, Germany, Dec. 2016, pp. 789–815.

[GJW19] Q. Guo, T. Johansson, and P. S. Wagner. "A Key Recovery Reaction Attack on QC-MDPC". In: *IEEE Trans. Inf. Theory* 65.3 (2019), pp. 1845–1861.

[HGS99] C. Hall, I. Goldberg, and B. Schneier. "Reaction Attacks against several Public-Key Cryptosystems". In: *ICICS 99: 2nd International Conference on Information and Communication Security*. Ed. by V. Varadharajan and Y. Mu. Vol. 1726. Lecture Notes in Computer Science. Sydney, Australia: Springer, Heidelberg, Germany, Nov. 1999, pp. 2–12.

[HHK17] D. Hofheinz, K. Hövelmanns, and E. Kiltz. "A Modular Analysis of the Fujisaki-Okamoto Transformation". In: *TCC 2017: 15th Theory of Cryptography Conference, Part I*. Ed. by Y. Kalai and L. Reyzin. Vol. 10677. Lecture Notes in Computer Science. Baltimore, MD, USA: Springer, Heidelberg, Germany, Nov. 2017, pp. 341–371.

[HLS21] C. Hlauschek, N. Lahr, and R. L. Schröder. *On the Timing Leakage of the Deterministic Re-encryption in HQC KEM*. Cryptology ePrint Archive, Report 2021/1485, version 20211115:124514 (posted 1636980314 15-Nov-2021 12:45:14 UTC). `https://eprint.iacr.org/2021/1485/20211115:124514`. Aug. 2021.

[Hog15] M. Hogan. "Data flows and water woes: The Utah Data Center". In: *Big Data & Society* 2.2 (2015).

[HPA21] J. Howe, T. Prest, and D. Apon. "SoK: How (not) to Design and Implement Post-quantum Cryptography". In: *Topics in Cryptology – CT-RSA 2021*. Ed. by K. G. Paterson. Cham: Springer International Publishing, 2021, pp. 444–477.

[Kal20] G. Kalai. "The Argument against Quantum Computers, the Quantum Laws of Nature, and Google's Supremacy Claims". In: *CoRR* abs/2008.05188 (2020).

[Kau+16] T. Kaufmann, H. Pelletier, S. Vaudenay, and K. Villegas. “When Constant-Time Source Yields Variable-Time Binary: Exploiting Curve25519-donna Built with MSVC 2015”. In: *Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings*. Ed. by S. Foresti and G. Persiano. Vol. 10052. Lecture Notes in Computer Science. 2016, pp. 573–582.

[KJJ99] P. Kocher, J. Jaffe, and B. Jun. “Differential Power Analysis”. In: *CRYPTO 1999*. Boston, MA: Springer US, 1999, pp. 388–397.

[Knu97] D. E. Knuth. *The art of computer programming, Volume I: Fundamental Algorithms, 3rd Edition*. Addison-Wesley, 1997.

[Koc96] P. C. Kocher. “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”. In: *Advances in Cryptology – CRYPTO’96*. Ed. by N. Koblitz. Vol. 1109. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1996, pp. 104–113.

[Lu+19] X. Lu et al. *LAC*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`. National Institute of Standards and Technology, 2019.

[Mer+21] R. Merget, M. Brinkmann, N. Aviram, J. Somorovsky, J. Mittmann, and J. Schwenk. “Raccoon Attack: Finding and Exploiting Most-Significant-Bit-Oracles in TLS-DH(E)”. In: *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*. Ed. by M. Bailey and R. Greenstadt. USENIX Association, 2021, pp. 213–230.

[Mis+13] R. Misoczki, J. Tillich, N. Sendrier, and P. S. L. M. Barreto. “MDPC-McEliece: New McEliece variants from Moderate Density Parity-Check codes”. In: *Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, July 7-12, 2013*. IEEE, 2013, pp. 2069–2073.

[Mog+20] D. Moghimi, B. Sunar, T. Eisenbarth, and N. Heninger. “TPM-FAIL: TPM meets Timing and Lattice Attacks”. In: *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*. Ed. by S. Capkun and F. Roesner. USENIX Association, 2020, pp. 2057–2073.

[Moo+20] D. Moody et al. *Status report on the second round of the NIST post-quantum cryptography standardization process*. Tech. rep. Gaithersburg, MD: National Institute of Standards and Technology, July 2020, pp. 1–27.

[Mos17] M. Mosca. *The Quantum Threat to Cybersecurity (for CxOs)*. Presented at the 5th ETSI-IQC Workshop on Quantum-Safe Cryptography. available at `https://docbox.etsi.org/Workshop/2017/201709_ETSI_IQC_QUANTUMSAFE/EXECUTIVE_TRACK/UNIofWATERLOO_MOSCA.pdf`. 2017.

[MOV97] A. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[Nae+20] M. Naehrig et al. *FrodoKEM*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`. National Institute of Standards and Technology, 2020.

[NJW18] A. Nilsson, T. Johansson, and P. S. Wagner. "Error Amplification in Code-based Cryptography". In: *IACR TCHES* 2019.1 (2018). `https://tches.iacr.org/index.php/TCHES/article/view/7340`, pp. 238–258.

[Pao10] G. Paoloni. *How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures*. Tech. rep. available at: `https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf`. 2010.

[PT19] T. B. Paiva and R. Terada. "A Timing Attack on the HQC Encryption Scheme". In: *SAC 2019*. Ed. by K. G. Paterson and D. Stebila. Vol. 11959. LNCS. Springer, Heidelberg, Aug. 2019, pp. 551–573.

[Qua20] QuantumRISC. *QuantumRISC — Next Generation Cryptography for Embedded Systems*. 2020.

[Rav+04] S. Ravi, P. C. Kocher, R. B. Lee, G. McGraw, and A. Raghunathan. "Security as a new dimension in embedded system design". In: *Proceedings of the 41th Design Automation Conference, DAC 2004, San Diego, CA, USA, June 7-11, 2004*. Ed. by S. Malik, L. Fix, and A. B. Kahng. ACM, 2004, pp. 753–760.

[Sch21] L. Schröder. "A Novel Timing Side-Channel Assisted Key-Recovery Attack Against HQC". `https://doi.org/10.34726/hss.2022.91042`. MA thesis. TU Darmstadt/TU Wien, 2021.

[Sen21] N. Sendrier. *Secure Sampling of Constant-Weight Words – Application to BIKE*. Cryptology ePrint Archive, Report 2021/1631, 20211217:142141 (posted 1639750901 17-Dec-2021 14:21:41 UTC). `https://eprint.iacr.org/2021/1631/20211217:142141`. Dec. 2021.

[SM16] D. Stebila and M. Mosca. "Post-quantum Key Exchange for the Internet and the Open Quantum Safe Project". In: *SAC 2016*. Ed. by R. Avanzi and H. M. Heys. Vol. 10532. LNCS. Springer, Heidelberg, Aug. 2016, pp. 14–37.

[SN16] N. I. of Standards and T. (NIST). *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*. available at: `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf`. 2016.

[Uen+22] R. Ueno, K. Xagawa, Y. Tanaka, A. Ito, J. Takahashi, and N. Homma. "Curse of Re-encryption: A Generic Power/EM Analysis on Post-Quantum KEMs". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022.1 (2022), pp. 296–322.

[van+20] T. van Goethem, C. Pöpper, W. Joosen, and M. Vanhoef. "Timeless Timing Attacks: Exploiting Concurrency to Leak Secrets over Remote Connections". In: *USENIX Security 2020*. Ed. by S. Capkun and F. Roesner. USENIX Association, Aug. 2020, pp. 1985–2002.

[Waf+19] G. Wafo-Tapa, S. Bettaieb, L. Bidoux, P. Gaborit, and E. Marcatel. *A Practicable Timing Attack Against HQC and its Countermeasure*. Cryptology ePrint Archive, Report 2019/909. `https://eprint.iacr.org/2019/909`. 2019.

[Wel47] B. L. Welch. "The generalisation of student's problems when several different population variances are involved". In: *Biometrika* 34.1-2 (Jan. 1947), pp. 28–35.

[WSN18] W. Wang, J. Szefer, and R. Niederhagen. "FPGA-Based Niederreiter Cryptosystem Using Binary Goppa Codes". In: *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*. Ed. by T. Lange and R. Steinwandt. Springer, Heidelberg, 2018, pp. 77–98.

[YZ17] Y. Yu and J. Zhang. *Lepton*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`. National Institute of Standards and Technology, 2017.

SCA-LDPC: A Code-Based Framework for Key-Recovery Side-Channel Attacks on Post-Quantum Encryption Schemes

Abstract

Whereas theoretical attacks on standardized crypto primitives rarely lead to actual practical attacks, the situation is different for side-channel attacks. Improvements in the performance of side-channel attacks are of utmost importance.

In this paper, we propose a framework to be used in key-recovery side-channel attacks on CCA-secure post-quantum encryption schemes. The basic idea is to construct chosen ciphertext queries to a plaintext checking oracle that collects information on a set of secret variables in a single query. Then a large number of such queries is considered, each related to a different set of secret variables, and they are modeled as a low-density parity-check code (LDPC code). Secret variables are finally determined through efficient iterative decoding methods, such as belief propagation, using soft information. The utilization of LDPC codes offers efficient decoding, source compression, and error correction benefits. It has been demonstrated that this approach provides significant improvements compared to previous work by reducing the required number of queries, such as the number of traces in a power attack.

The framework is demonstrated and implemented in two different cases. On one hand, we attack implementations of HQC in a timing attack, lowering the

Q. Guo, D. Nabokov, A. Nilsson, and T. Johansson. *SCA-LDPC: A Code-Based Framework for Key-Recovery Side-Channel Attacks on Post-Quantum Encryption Schemes*. Submission Pending. 2023 number of required traces considerably compared to attacks in previous work. On the other hand, we describe and implement a full attack on a masked implementation of Kyber using power analysis. Using the ChipWhisperer evaluation platform, our real-world attacks recover the long-term secret key of a first-order masked implementation of Kyber-768 with an average of only 12 power traces.

1 Introduction

NIST [NIS18] is running a standardization process (referred to as the NIST PQ project) for post-quantum public-key cryptographic algorithms (PQC schemes), which are supposed to be secure even against attacks from quantum computers. This is not the case for most public-key algorithms in use today [Sho94]. The project started in 2017 and just recently the first choices for standardization were announced. The project is ongoing and round 4 will involve a further examination of additional schemes. All of the schemes in the NIST PQ project are based on a variety of hard problems that are believed to be intractable for quantum computers, and many of them can be categorized as either Public-key Encryption (PKE) or Key Encapsulation Mechanisms (KEMs). These PKE/KEM schemes are based on either the Learning with Errors (LWE) problem as introduced by Regev [Reg05] in 2005 or on code-based problems, initiated in [McE78].

Two such schemes will be considered in this paper. One is CRYSTALS-Kyber [Sch+20], selected by NIST as the candidate for standardization for KEMs. The security of Kyber is based on the Module LWE problem and has strong confidence in its theoretical security, while also offering a good performance. The other scheme is HQC [Agu+20], a code-based round 4 candidate. Other code-based round 4 candidates are BIKE [Ara+20] and Classic McEliece [Alb+20]. NIST has stated that one of the schemes HQC or BIKE may be standardized.

LWE- or code-based PKE/KEMs are usually built to be secure against chosen plaintext attacks (IND-CPA secure) and then transformed to be secure against adaptive chosen ciphertext attacks (IND-CCA secure) by applying some CCA conversion method, such as the Fujisaki-Okamoto (FO) transform. The FO transform involves a re-encryption after decryption, which enables the detection of invalid ciphertexts and correspondingly return failure. Invalid chosen ciphertexts that are not proper encryptions of a message will almost always be rejected by the decryption/decapsulation.

Side-Channel Attacks (SCA) were introduced by Kocher [Koc96] and are a separate area of research today. For PQC schemes, it is a major concern and NIST also in the later rounds encouraged more research on the security of PQC schemes against side-channel cryptanalysis. In relation to this, there has been great research interest in developing new side-channel attacks on all relevant NIST candidates as well as studying efficient side-channel protection techniques.

There are many different approaches to SCA on PQC schemes. Following previous work, we may roughly classify attacks into two main categories. The first category includes attacks that require either a single trace or at least only few traces to perform key recovery or message recovery and targets very precise leakages in an implementation. The second category includes attacks of a more generic type, exploiting arbitrary leakages in the implementation of the algorithm, but typically requiring the collection of many traces in the attack phase. These more generic attacks are modeled by instantiating a side-channel oracle for chosen ciphertexts. The oracle is explained in more detail in the following.

1.1 Related works

Key-recovery chosen-ciphertext side-channel attacks (KR-CCA-SCA) are attacks where the adversary recovers the secret key in the scheme by using chosen ciphertext calls to the decryption or decapsulation algorithm and getting measurement data from some side-channel.

KR-CCA-SCA attacks on PQC encryption schemes are a well-established research field, as evidenced by numerous publications [DAn+19; Rav+20; Ngo+21; Xu+22; GJN20; Guo+22; Ham+21; GJJ22; Sch+22; GLG22; Rav+22; She+22]. These attacks can be classified depending on where the information leakages are detected. The first type of KR-CCA-SCAs [GJN20; Guo+22; Rav+20] exploits leakages from the two added procedures, the re-encryption and ciphertext comparison, of the FO transform, since these two components in the FO transform depend on the decrypted message vector. There are also KR-CCA-SCAs [Ngo+21; Ham+21; GJJ22; Sch+22; GLG22] that exploit side-channel leakages from the CPA-secure decryption, where parts of the decryption procedure will directly use the secret key.

In [RR], Ravi et al. classified side-channel-assisted CCA attacks on lattice-based KEMs into three main categories, plaintext-checking (PC) oracle based attacks [DAn+19; Rav+20], decryption-failure (DF) oracle based attacks [GJN20], and full-domain (FD) oracle based attacks [Ngo+21; Xu+22]. The classification depends on what kind of answer the oracle gives. In a DF oracle, the oracle answer is simply whether the chosen ciphertext decodes/decrypts to a valid message or not. On the other hand, a PC oracle and an FD oracle require message recovery before key recovery can take place. In a PC oracle, the response of the oracle is whether the chosen ciphertext results in a specific given message upon decryption. In an FD Oracle, the oracle returns the full message that has been decrypted. As a result, in a PC oracle based attack, it is possible to recover a maximum of one bit of secret information from a single side-channel measurement; however, if the message is of m-bit length (where m is 256 for Kyber), it is possible to recover m bits of secret information with a FD oracle based attack.

Recently, Tanaka et al. in [Tan+22] and Rajendran et al. in [Raj+22] have independently proposed a new type of oracle called multi-values PC oracle. This oracle can extract 8-12 bits of information from a single decapsulation oracle call through multi-class classification. The multi-values PC oracle can be considered as a compromise between the PC oracle and the FD oracle, although it is still much less efficient than the latter.

For general PQC schemes, Ueno et al. [Uen+22] have shown that all round-3 NIST KEM candidates except for Classic McEliece are vulnerable to KR-CCA-SCAs. However, it was later established in [Sch+22] that the attack detailed in [Uen+22] is only applicable to earlier versions of the HQC proposal and not to the recent Reed-Muller-Reed-Solomon (RMRS) version. Schamberger et al. in [Sch+22] and Goy et al. in [GLG22] very recently proposed new power side-channel attacks on the RMRS version of the HQC scheme, but their attack only applied to power analysis with leakages from the CPA decryption. In [Guo+22] a generic PC oracle based attack on the RMRS version of the HQC scheme has been proposed, presented in the format of timing attacks.

One central problem in KR-CCA-SCAs is identifying a generic approach to optimize the selection of chosen ciphertexts, in order to efficiently extract information from side-channel measurements. The main obstacles arise from two primary sources: (1) the inaccuracies that may occur in the construction of oracles, particularly with the most powerful FD oracles, and (2) the non-uniform distribution from which secret symbols are generated. To overcome these challenges, it is necessary to incorporate concepts from coding theory, particularly in the areas of source coding and error correction. Several early research efforts are made to address these challenges, as documented in [Ngo+21; Qin+21; She+22]. But the existing solutions are limited in scope, either because they are restricted to a specific oracle or because they are applicable only to particular types of side-channel leakages. Finally, there is ample room for improvement in terms of attack efficiency.

1.2 Contributions

In this paper, we propose a framework named SCA-LDPC to improve the key-recovery side-channel attacks on CCA-secure PQC encryption schemes. The basic idea is to construct chosen ciphertext queries to an oracle that collects information on a set of secret variables in a single query. Then a large number of such queries are considered, each related to a different set of secret variables, and they are modeled as a low-density parity-check code (LDPC code). The secret variables are then determined through efficient iterative decoding methods, such as belief propagation (BP), using soft information.

New concepts. The concept of designing chosen-ciphertexts to gather side-channel information in a linear parity check is a fresh and innovative approach. This approach has the potential to provide both source compression and error correction simultaneously. The reason for this is that the combination of multiple secret entries is typically more closely aligned with the uniform distribution, which allows for more effective extraction of information from a single side-channel measurement. This source compression gain can result in a substantial improvement for

HQC where secret symbols have an extremely low entropy, as well as a noticeable improvement for lattice-based schemes. The error correction gain is realized through the utilization of linear parity checks, which enable the utilization of correctly recovered coefficients to rectify incorrect decisions. The implementation of these linear checks in the form of an LDPC code was selected due to its efficient decoding capabilities and its well-known near-optimal performance from an information-theoretical perspective.

The new framework has significantly transformed the design philosophy of prior methods for source compression and error correction, as documented in [Ngo+21; Qin+21; She+22]. The previously proposed methods aimed to achieve full key recovery with higher accuracy by introducing additional measurements for each individual secret symbol, thereby increasing the success rate of symbol recovery. The new framework, however, proposes a novel approach by allowing for fewer measurements on the secret symbols, leading to a higher level of symbol-level errors, which are subsequently corrected by the specially designed LDPC codes through inter-symbol parity checks.

We emphasize that the framework is generic in nature and can be applied to both code-based and lattice-based schemes, across adaptive and non-adaptive attack models, and in a multitude of side-channel leakage scenarios, including timing, cache-timing, power, and electromagnetic leakages. To demonstrate the applicability of the framework, we have instantiated it in two relevant applications: an adaptive timing attack on an HQC implementation with PC oracles, and a non-adaptive power attack on a Kyber implementation with FD oracles. The choice of Kyber and HQC as the primary targets was motivated by their significance, with Kyber being selected as the primary KEM/PKE algorithm for standardization by NIST and HQC still being considered for standardization at the end of round-4.

New results. We list the contributions of the paper in the following.

- We introduce a code design method in designing capacity-approaching LDPC /QC-LDPC codes over binary and non-binary alphabets. Our method establishes a relationship between oracle calls and parity checks in the LDPC code, leading to substantial improvements over previous methods in both noiseless and noisy real-world scenarios. The prior improvement is primarily attributed to source compression, while the latter improvement is the result of a combination of source compression and error correction.

- We simulate the performance with different noise levels and characterize the performance of the new approach through a simulation method. The simulated gains are substantial. For example, when the oracle accuracy is 100%, as is the case for the key misuse oracle or an oracle constructed from highly reliable side-channels such as cache-timing leaks on an Intel-SGX platform [HSC], we can recover the secret key of `hqc-128` with approximately 9,000 traces in the PC oracle. Using the same oracle setting as the ideal oracle in [Guo+22], we have achieved an improvement factor of 86.6, as we only need about 10,000 traces, compared to the 866,000 traces reported in [Guo+22]. This significant improvement is due to the fact that the HQC secret entries are sampled from a distribution with extremely low entropy and the previously known methods (e.g., in [Guo+22]) ignore the potential source compression gain. In the scenario of perfect FD oracles, the number of traces required to recover Kyber-768 is only 7, which meets the Shannon lower bound.

- We perform actual attacks on two target algorithms, Kyber and HQC, in real-world scenarios. The results of our study demonstrate a close alignment, or even an improvement, of the real attack performance when compared to the simulation results. The first attack is a full power analysis on a masked implementation of Kyber-768. The attack was carried out using the ChipWhisperer framework on the open-source mkm4 library in [Hei+22] with the profiling and attack phases performed on two distinct boards, both equipped with ARM-Cortex-M4 CPUs. In the real-world scenario, we obtained FD oracles with varying accuracy levels based on their positions. The average accuracy was estimated to be approximately 95%. The full secret key was successfully recovered with an average of 12 traces. In comparison, the simulation required roughly 17 traces for the same oracle accuracy. The better performance in the real-world scenario can be attributed to the availability of soft information and the possibility of some secret symbols having a high accuracy since they are related to a high-accuracy oracle, which in turn helps in the correct decoding of other positions through the parity checks of the specially designed LDPC codes.

  The second attack is a full timing attack simulation on HQC, validated with a real-world timing oracle. The real-life attack performance highly depends on the targeted platform. On our laptop with an Intel Core i5 CPU, we can achieve full key recovery against hqc-128 with 2^{18} decapsulation calls.

- The software for attack and simulation will be made open-source.[1]

It is important to note that, in accordance with previous research, our approach focuses on recovering the entire secret vector through side-channel leakages. The sample complexity can be reduced by performing additional post-processing procedures, such as information set decoding and lattice reduction [Dac+20], to recover a portion of the secret entries. The specific reduction in the number of traces depends on the permissible amount of computation for post-processing.

Comparison with previous studies in [Ngo+21; Qin+21; She+22]. In [Qin+21], Qin et al. presented an efficient PC oracle based attack on lattice-based schemes

[1] https://github.com/atneit/SCA-LDPC

by adaptively choosing a new ciphertext for decryption based on the side-channel information obtained from previous power/electromagnetic measurements. Their approach is similar to the well-known Huffman coding method and can result in a source compression gain. Shen et al. in [She+22] further extended this work by proposing a detection coding method to identify incorrectly recovered positions and to send additional measurements for those secret positions. Note that these studies are limited to PC oracle based attacks on lattice-based schemes and operate in adaptive mode, resulting in a more restrictive attack model and lower efficiency compared to other attacks based on more powerful oracles. For example, when the oracle accuracy is 95%, it was reported in [She+22] that 3874 traces are required to attack the Kyber-512 scheme; in contrast, the new attack based on the FD oracle presented in this paper only requires 12 traces in a real attack (or 17 traces in simulation) to attack the Kyber-768 scheme. Furthermore, from a viewpoint of information theory, LDPC codes are attractive due to their near-optimal performance. As a result, they are expected to provide improved performance in scenarios where the oracle accuracy is low, compared to the detection codes proposed in [She+22].

A relevant study [Ngo+21] has proposed the extended Hamming coding method to enhance the FD oracle based power attack on the masked Saber, a round-3 NIST PQ KEM candidate, in the non-adaptive attack model. However, this approach does not provide any source coding gain and its error correction is limited to an inner-symbol style, resulting in a lack of inter-symbol connections and a less potent error correction mechanism. A more detailed comparison of our work with [Ngo+21] can be found in Section 6.3.

Differences from SASCA. Veyrat-Charvillon et al. in [VGS14] utilized iterative decoding to introduce a powerful side-channel attack method named soft-analytic side-channel attacks (SASCA). Their objective was to efficiently exploit the leakage of intermediate variables. They observed that secret variables and intermediate variables are connected by computation functions/gates; thus, the leakages from variables closely connected to the secret variables can be extracted via iterative decoding on the codes built from these connections. It should be noted that the computation process defines the underlying codes, which are generally non-linear, and that better cipher design to resist SASCA entails generating a computation process that leads to a code with poor decoding performance. A coding-theoretical treatment on SASCA is presented in [Guo+20]. SASCA has been employed for attacking lattice-based scenarios by exploiting leakages from the number-theoretical transform (NTT) of secret polynomials (e.g., in [PPM17; Ham+21]).

While both attacks use iterative coding, the new SCA-LDPC attack differs from SASCA in several ways. First, the applicability of SASCA extends to both symmetric and asymmetric cryptography; in contrast, the SCA-LDPC attacks fall into the category of PC oracle (and its variants) based KR-CCA-SCAs for lattice-based and code-based systems, presenting unique advantages for these types of systems. For instance, similar to the previous PC oracle based KR-CCA-SCAs, the SCA-LDPC attacks can efficiently exploit leakages from the FO transform or other procedures in the latter stages of the decapsulation algorithm, for key recovery; SASCA can be more efficient in extracting information from leakages when the secret key has been used directly or within a few computational steps before, as seen in the NTT attack instances. Moreover, SCA-LDPC attacks are capable of exploiting different types of leakages, including timing-related ones that are hardly exploitable through SASCA. The second primary distinction is that SASCA builds its code using pre-existing connections formed by the intrinsic structure of the cipher or implementation, whereas SCA-LDPC provides the attacker with greater flexibility in choosing ciphertexts to create new parity check variables and establish new connections, ultimately constructing a linear code with near-optimal decoding efficiency. Last, SCA-LDPC generates linear parity checks, while SASCA usually involves non-linear functions.

1.3 Relations to very recent work

The first draft of the work was finalized in October 2022. In December 2022, two relevant studies [Bac+22; DNG22] on the topic of FD-oracle based attacks on masked implementations of Kyber were made available on the IACR ePrint website. These works extend the applications of the method proposed in [Ngo+21], and as such, differ from the direction taken by our framework in its design towards a more efficient information extraction. These two works offer no source compression gains and the error correction approach is similar to that proposed in [Ngo+21]. For example, in [Bac+22], the authors constructed a masked and shuffled implementation based on the first-order masked implementation in [Hei+22] and reported a minimum required number of traces of 38016. While our reported number is 12, a direct comparison between these two attack instances is invalid as the targeted implementations differ. In [DNG22], Dubrova et al. presented a high probability of success in the recovery of message bits from a masked implementation of Kyber up to the fifth order. Their results suggest that our SCA-LDPC attack framework may perform effectively even against masked implementations with a much higher order. Additionally, the authors identified stronger leakage points from the function `masked_poly_frommsg` compared to `masked_poly_tomsg` found in the initial version of our paper. As a result, we have revised our approach to utilize the new leakage points reported in [DNG22] for a more efficient attack.

In January 2023, Huang et al. presented the first KR-CCA-SCA in the context of cache timing attacks in [HSC]. Their results include a novel approach for using a PC oracle for the KR-CCA-SCA attack on the RMRS version of the HQC scheme. The accuracy of the PC oracle in the cache scenario is high, with a reported rate of almost 100%. However, their method does not offer any source compression or error correction gains and requires a significantly higher number of oracle calls compared to our SCA-LDPC framework (i.e., 53857 vs. 9000). The potential to integrate their attack concept with our SCA-LDPC framework for source compression and error correction is an area of future investigation.

1.4 Organization

The remaining parts of the paper are organized as follows. In Section 2, we present the necessary background information. In Section 3, we present a general description of the new attacking framework. We apply the new attack ideas towards Kyber and HQC, in Sections 4 and 5, respectively. Then, we present the extensive computer simulation results and real-world attacks in Section 6. We finally conclude the paper and present future directions in Section 7.

2 Preliminaries

We present the necessary background in this section. We first provide the employed notations and terminology in coding theory, followed by a description of the two KEM candidates, Kyber and HQC. Finally, we conclude this section by outlining the threat model.

2.1 Notations and coding terminology

Notation. For a finite set $\mathcal{I}$, the symbol $\#\{\mathcal{I}\}$ denotes the number of elements in $\mathcal{I}$. Let $\mathbb{F}_q$ be the finite field of size q, $\lceil x \rfloor$ the rounding function, and $\mathcal{H}$, $\mathcal{G}$, and $\mathcal{K}$ three cryptographic hash functions. The central binomial distribution B_μ outputs $\sum_{i=1}^{\mu}(a_i - b_i)$, where a_i and b_i are independently and uniformly randomly sampled from . The Bernoulli distribution Ber_η defines a random variable from , which is 1 with probability η and 0 otherwise. The notation $\mathbf{a} \xleftarrow{\$} \mathsf{U}$ denotes that the entries in $\mathbf{a}$ are randomly sampled from the distribution U, where $\mathbf{a}$ is a vector or polynomial. For a set $\mathcal{I}$, $\mathbf{a} \xleftarrow{\$} \mathcal{I}$ means that the entries in $\mathbf{a}$ are uniformly sampled from the set $\mathcal{I}$ at random. For a vector or polynomial $\mathbf{a}$, $\mathbf{a}[i]$ refers to the coefficient of $\mathbf{a}$ at the index i. The Shannon's binary entropy function of a random variable X is defined as $H(X) = -\sum_{x \in \mathcal{X}} \mathbf{Pr}[X = x] \log_2 \mathbf{Pr}[X = x]$.

Linear codes. The Hamming weight of a vector $\mathbf{x}$ is its number of non-zero elements, denoted by $w_{\mathrm{H}}(\mathbf{x})$. We define an $[n, k, d]_q$ linear code $\mathcal{C}$ as a linear subspace over $\mathbb{F}_q$ of length n, dimension k, and minimum distance d. Here minimum distance is defined as the minimum Hamming weight of its non-zero elements. Since a linear code $\mathcal{C}$ is a subspace, we can define it as the image of a matrix $\mathbf{G}$, called a *generator matrix*. We can also define the code $\mathcal{C}$ as the kernel of a matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$. Here $\mathbf{H}$ is called a parity-check matrix of $\mathcal{C}$.

LDPC codes. *Low-density parity-check* (LDPC) codes are linear codes with a sparse parity-check matrix first introduced in [Gal62]. LDPC codes can be considered sparse graph codes because they can be decoded efficiently using iterative decoding (such as belief propagation [Pea82]) on the *Tanner graph*, a bipartite graph with edges corresponding to non-zero elements in the parity-check matrix **H**.

Concatenated codes. Forney [For65] in 1965 firstly proposed the concatenated code construction approach of combining two simple codes called an *inner code* and an *outer code*, respectively, to achieve good error-correcting capability with reasonable decoding complexity. Let the inner code $\mathcal{C}_{in} : \mathcal{A}^k \rightarrow \mathcal{A}^n$, the outer code $\mathcal{C}_{out} : \mathcal{B}^K \rightarrow \mathcal{B}^N$, and $\#\{\mathcal{B}\} = \#\{\mathcal{A}\}^k$. The concatenated code is a code $\mathcal{C}_{con} : \mathcal{A}^{kK} \rightarrow \mathcal{A}^{nN}$. The key of the concatenated code construction method is that the decoding can be done sequentially by passing first the inner code decoder and then the outer code decoder. Typically in the inner code decoding, one can use a maximum-likelihood decoding approach, while the outer code allows efficient decoding in polynomial time (e.g. by employing an LDPC code).

2.2 Kyber

Kyber [Sch+20], the KEM version of the Cryptographic Suite for Algebraic Lattices (CRYSTALS), is based on the module Learning with Errors (MLWE) problem and has been solicited as the KEM/PKE standard in the NIST PQ project.

Kyber achieves the IND-CCA security through a tweaked Fujisaki-Okamoto transform [FO99] transforming an IND-CPA-secure PKE Kyber.CPAPKE to an IND-CCA-secure KEM Kyber.CCAKEM. The description algorithms of Kyber. CPAPKE and Kyber.CCAKEM can be found in [Sch+20]. We include a simplified description in Figs. 3 and 4 for completeness, where the implementation details with the Number Theoretical Transform (NTT) are omitted.

In the following, we define the compression function and the decompression function, i.e., $\mathbf{Comp}_q(x, d)$ and $\mathbf{Decomp}_q(x, d)$, respectively.

Definition 1. *The Compression function is defined as:* $\mathbb{Z}_q \rightarrow \mathbb{Z}_{2^d}$

$$\mathbf{Comp}_q(x, d) = \left\lceil \frac{2^d}{q} \cdot x \right\rfloor \pmod{2^d}. \tag{1}$$

Definition 2. *The Decompression function is defined as:* $\mathbb{Z}_{2^d} \rightarrow \mathbb{Z}_q$

$$\mathbf{Decomp}_q(x, d) = \left\lceil \frac{q}{2^d} \cdot x \right\rfloor. \tag{2}$$

The compression and decompression function can be done coefficient-wise if the input is a polynomial or a vector of polynomials $\mathbf{x} \in \mathcal{R}_q^d$. The procedure KDF(·) denotes a key-derivation function.

Table 1: Parameter sets for Kyber [Sch+20]

	n_{mod}	d	q	μ_1	μ_2	(d_u, d_v)
Kyber-512	256	2	3329	3	2	(10,4)
Kyber-768	256	3	3329	2	2	(10,4)
Kyber-1024	256	4	3329	2	2	(11,5)

The security parameter sets for the three versions of Kyber, Kyber-512, Kyber-768, and Kyber-1024 are shown in Table 1. In Kyber q is a prime 3329. Let $\mathcal{R}_q$ be a polynomial ring $\mathbb{F}_q[x]/(x^{256}+1)$. Let $\mathbf{H}_0$ be a *negacyclic* matrix from a vector $\mathbf{h}_0$, i.e. the first row is $\mathbf{h}_0$, subsequent rows are cyclically shifted, when the value is moved from the last column to the first one, it is multiplied by -1. Let d denote the rank of the module, set to be 2, 3, and 4, respectively, for Kyber-512, Kyber-768, and Kyber-1024. When sampling from central binomial distribution B_μ, Kyber also has two parameters (μ_1, μ_2), set to be $(3, 2)$ for Kyber-512 and $(2, 2)$ for Kyber-768 and Kyber-1024.

2.3 HQC

HQC (Hamming Quasi-Cyclic) [Agu+20] is one of the main code-based IND-CCA-secure KEMs in the NIST PQ project, which has advanced to the fourth round. Its security is based on the hardness of decoding a random quasi-cyclic code in the Hamming metric. In HQC, the base field is $\mathbb{F}_2$ and $\mathcal{R}_2$ denotes the polynomial ring $\mathbb{F}_2[x]/(x^n - 1)$. The multiplication of two polynomials $\mathbf{u}, \mathbf{v} \in \mathcal{R}_2$ can be represented as a vector and a *circulant matrix*, induced from a vector in $\mathbb{F}_2^n$. Given $\mathbf{y} = (y_1, y_2, \ldots, y_n) \in \mathbb{F}_2^n$, its corresponding circulant matrix is defined as

$$\mathbf{rot}(\mathbf{y}) = \begin{pmatrix} y_1 & y_n & \cdots & y_2 \\ y_2 & y_1 & \cdots & y_3 \\ \vdots & \vdots & \ddots & \vdots \\ y_n & y_{n-1} & \cdots & y_1 \end{pmatrix}.$$

We can write the multiplication of $\mathbf{uv}$ as $\mathbf{u} \cdot \mathbf{rot}(\mathbf{v})^{\mathrm{T}}$ or $\mathbf{v} \cdot \mathbf{rot}(\mathbf{u})^{\mathrm{T}}$. The transpose of the circulant matrix is the counterpart of the negacyclic matrix.

The detailed description of the IND-CPA-secure PKE version of HQC and the IND-CCA-secure KEM version can be found in the HQC reference document [Agu+20]. We also list them in Figs. 5 and 6 for completeness. The procedure $\mathsf{KeyGen}(\cdot)$ randomly generates two private vectors $\mathbf{x}, \mathbf{y} \in \mathcal{R}_2$ with a low Hamming weight w as the private key. It also generates a random public vector $\mathbf{h} \in \mathcal{R}_2$, computes $\mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y}$, and returns $(\mathbf{h}, \mathbf{s})$ as the public key. The scheme employs a linear code $\mathcal{C}$ with a generator matrix $\mathbf{G}$ and generates noise $\mathbf{e}, \mathbf{r}_1, \mathbf{r}_2 \in \mathcal{R}_2$ with low Hamming weight in the encryption. The encryption

Table 2: The HQC parameter sets [Agu+20]. The inner code is the duplicated Reed-Muller code defined by the first-order $[128, 8, 64]_2$ Reed-Muller code.

	RS-S			Duplicated RM						
Instance	n_1	k	d_{RS}	Mult.	n_2	d_{RM}	$n_1 n_2$	n	ω	$\omega_r = \omega_e$
hqc-128	46	16	31	3	384	192	17,664	17,669	66	75
hqc-192	56	24	33	5	640	320	35,840	35,851	100	114
hqc-256	90	32	49	5	640	320	57,600	57,637	131	149

function computes $\mathbf{u} = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2$ and $\mathbf{v} = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}$ and returns $(\mathbf{u}, \mathbf{v})$ as the ciphertext. In decryption, the secret vector $\mathbf{y}$ is an input and it computes

$$\mathbf{v} - \mathbf{u} \cdot \mathbf{y} = \mathbf{mG} + \underbrace{\mathbf{s} \cdot \mathbf{r}_2 - \mathbf{u} \cdot \mathbf{y} + \mathbf{e}}_{\hat{\mathbf{e}}} . \tag{3}$$

Since $w_{\text{H}}(\hat{\mathbf{e}})$ is small, the decryption function inputs $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ to the decoder of $\mathcal{C}$ and can succeed with high probability.

The parameter sets of HQC are shown in Table 2. In the recent version published in June 2021, HQC employs a concatenation of outer $[n_1, k_1, n_1 - k_1 + 1]_{256}$ Reed-Solomon (RS) codes and inner duplicated Reed-Muller (RM) codes built from the first-order $[128, 8, 64]_2$ Reed-Muller code. The encoding procedure first encodes a message $\mathbf{m} \in \mathbb{F}_2^{8k_1}$ to a codeword $\hat{\mathbf{m}} \in \mathbb{F}_{2^8}^{n_1}$ of the employed shortened Reed-Solomon codes. It then maps each byte of $\hat{\mathbf{m}}$ to a codeword of the first-order RM and repeats the RM codeword for 3 or 5 times depending on the security level to obtain a duplicated RM codeword in $\mathbb{F}_2^{n_2}$. In summary, we employ a linear code $\mathbf{mG} \in \mathbb{F}_2^{n_1 n_2}$. The HQC proposal makes all computations in the ambient space $\mathbb{F}_2^n$ and truncates the remaining $n - n_1 n_2$ useless bits.

The IND-CCA security of the KEM version of HQC is achieved by the Hofheinz-Hövelmanns-Kiltz (HHK) transform [HHK17].

2.4 Threat model

We consider a side-channel-assisted chosen-ciphertext attack on a KEM's decapsulation algorithm, where the attacker selects ciphertexts and observes specific side-channel data, such as timing [GJN20], cache-timing [HSC], or power/electromagnetic leakages [Rav+20], from the targeted device, which can be a high-end CPU, low-end CPU (e.g., ARM cortex-M4), or hardware device.

Specifically, we assume that a communication party Alice is using her device for key establishment. An adversary called Malory sends selected ciphertexts to Alice to recover Alice's long-term secret key. Alice runs the decapsulation algorithm and Malory will fail if the used KEM algorithm is IND-CCA secure. However, the designed side-channel-assisted CCAs can make the attack successful after a few such attempts, using the observed side-channel leakages.

This side-channel-assisted CCA attack model is well-established – it is stated in [Uen+22] that all the NIST round-3 KEM candidates except for Classic McEliece are vulnerable to such attacks exploring leakages from FO transform. The basic idea is to construct a plaintext-checking(PC) oracle outputting whether $\mathbf{Dec}(c') \overset{?}{=} m$, where c' is the chosen ciphertext and m is a message vector.

Finally, the attacker recovers the long-term secret keys based on the output of the PC oracle. Since the PC oracle is generally built from measurements of side-channel leakages, it cannot be 100% correct, in practice. We denote the accuracy of the constructed PC oracle ρ, i.e., the oracle outputs the right decision with probability ρ and the wrong one with probability $1 - \rho$.

Note that we are discussing general methods for near-optimal CCA SCAs. This new coding-theoretical approach for reduced sample (trace) complexity can be applied in various side-channel attacks on various platforms, while the starting oracle accuracy ρ can be different. A PC oracle with 100% correctness ($\rho = 1$) can also be connected to a key misuse attack model, as described in [Qin+21].

Profiled power/EM attacks. Specific to power/EM attacks, we mainly consider a profiled setting that the adversary has a similar but different device to perform training activities. Though the adversary has no access to the secret key in the targeted device, the secret key in the training device can be freely set. We can also apply the new idea to non-profiled attacks that can build the required abstract oracles online, but the sample complexity analysis will be different.

Comparison with the adaptive model in power/EM attacks. The studies [Qin+21; Raj+22; She+22] proposed efficient adaptive KR-CCA-SCAs on lattice-based proposals. This attack model allows the adversary to select a new chosen ciphertext based on information obtained from previous power/EM traces, which can be employed for source coding on secret coefficients. This approach can result in reduced sample complexity close to the lower Huffman or Shannon bounds. However, this attack model is strong for many practical (say IoT) applications since the adversary needs to have good connections with the device measuring the power/EM leakages and good computation capability to instantly process the obtained traces. We highlight that our new SCA-LDPC attack framework eliminates the requirement and offers source coding gain in a non-adaptive attack model.

3 General Description of the SCA-LDPC Attack Framework

This section presents a new idea of incorporating LDPC codes and soft information to design chosen ciphertexts and improve previously established KR-CCA-SCAs for CCA-secure post-quantum Key Encapsulation Mechanisms (KEMs) and encryption schemes. We propose a novel technique to extract, from a single side-channel measurement, information regarding a low-weight parity check of the secret coefficients, as opposed to information regarding a single coefficient in previous methods. The sparse system is then solved using iterative decoding methods, such as belief propagation. This new approach enables the attainment of both source compression benefits and error correction advantages. This is due to the combination of several secret coefficients, which leads to a more uniform extraction of information from a single trace. Additionally, the correct recovery of coefficients facilitates the correction of erroneous decisions through spare parity-check relations. The adoption of this new method significantly reduces the number of necessary side-channel measurements. We call the new attack strategy a framework as it is generic and can be applied to both code-based and lattice-based schemes, in a multitude of side-channel leakage scenarios including timing, cache-timing, power, and electromagnetic leakages.

We start this section by assuming the availability of a well-designed LDPC code with specific dimensions and proceed to explain its utilization for improved side-channel information extraction. We then in Section 3.2 present a simple method for constructing such linear codes, the effectiveness of which will be demonstrated through experiments in Section 6. In addition, we broaden the framework by introducing a concatenated construction, where the LDPC codes are utilized as the outer code. This construction is particularly efficient for lattice-based schemes that feature a large alphabet size or for scenarios where the accuracy of the oracle constructed from side-channel measurements is limited.

3.1 New attack idea

Given a good linear code with a sparse parity-check matrix $\mathbf{H}_{r\times n}$, there are $k = n - r$ secret positions to recover. In lattice-based and code-based KEM proposals, the value k is usually divided into b blocks, each of which has the size of k/b. We add the constraint that $\mathbf{H}$ should have the form of

$$\mathbf{H} = \left[\mathbf{H}_{r\times k} | - \mathbf{I}_{r\times r}\right].$$

Our goal is to recover the first k secret entries s_i. One parity-check equation, i.e., one row in the parity-check matrix $\mathbf{H}$, will introduce one check variable c_i for $i \in \{k+1, \ldots, k+r\}$. We can rewrite each parity-check equation as $c_i = \sum_{j\in\mathcal{I}} s_j$ and the size of $\#\{\mathcal{I}\}$ is small since the matrix $\mathbf{H}$ is of low density.

The secret entries s_i are typically generated according to a certain secret distribution. For example, in the lattice-based scheme Kyber, the secret entries are generated from the central binomial distribution B_μ; in the code-based scheme HQC, the secret vector is very sparse and each secret entry can be viewed as a Bernoulli variable Ber_η, where η is a small positive number. The secret distribution can be utilized as the prior information for s_i. Moreover, additional information can be obtained through the implementation of side-channel measurements

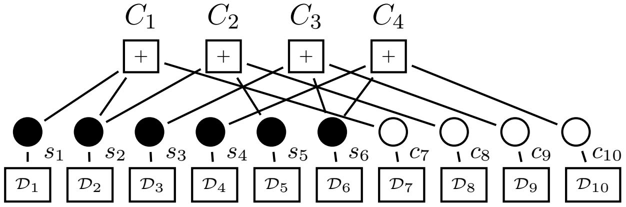


Figure 1: The Tanner graph explanation.

of s_i, which subsequently updates the relevant distribution. This approach is particularly beneficial in lattice-based scenarios. We then design new ciphertexts to obtain side-channel leakages of sparse linear combination c_i of s_j for $j \in \mathcal{I}$. The side channel information could reveal an empirical probability of c_i. The problem of recovering all s_i for $i \in \{1, \ldots, k\}$ is transformed into a coding problem through a noisy discrete channel. Note that the design method for ciphertexts that can reveal partial information of c_i, is unique to each proposed scheme and differs between lattice-based and code-based schemes. This ciphertext design is one main technical challenge in the proposed attack framework.

Explanation. The attack idea is illustrated in Fig. 1. We assume that 6 secret coefficients or variables s_i for $1 \leq i \leq 6$ need to be recovered. For each s_i, we can use the a priori distribution (e.g., in the HQC case), or we have more traces or oracle calls to get a better knowledge of its distribution (e.g., in the Kyber case). We show 4 parity checks in this example, and each check connects to a new variable v_i. From side-channel measurements or oracle calls, we got additional information about these variables. Thus, we could assign the corresponding distribution to these variables and build a Tanner graph as in Fig. 1. With this sparse bipartite Tanner graph, we perform iterative decoding to recover the desired secret coefficients s_i for $1 \leq i \leq 6$.

The gain of using LDPC codes. It is essential to select a sparse graph code that facilitates efficient decoding and renders the key recovery procedure computationally feasible. Therefore, it is natural to examine LDPC codes that have favorable characteristics from an information-theoretic point of view. We introduce the variables c_i, which are sparse linear combinations of the secret coefficients s_j, thereby facilitating a more efficient extraction of information from a single side-channel measurement. This is due to the fact that the distribution of c_i is typically closer to a uniform distribution compared to the distribution of s_i, resulting in substantial source compression gains, particularly in the case of HQC and to a significant extent in Kyber. Further discussions regarding these source compression gains will be presented in Section 6. Finally, LDPC codes can offer close to optimal error correction performance, rendering the attack framework efficient in terms of the number of side-channel measurements required, even when the oracle constructed from side-channel leakages is highly inaccurate.

Example 2 (The source compression gain for $\mathsf{hqc\text{-}128}$). *In $\mathsf{hqc\text{-}128}$, the length of* $\mathbf{y}$ *is* $n = 17669$ *and the Hamming weight of* $\mathbf{y}$ *is* $w_H(\mathbf{y}) = 66$. *Hence, we can approximate each position of* $\mathbf{y}$ *as a Bernoulli distribution* Ber_η, *where* $\eta \approx 0.0037$. *Assume that we have a perfect oracle to inform us of the value of one position from one oracle call. If we try to recover a bit in* $\mathbf{y}$ *by one oracle call, with Shannon's binary entropy function, the obtained information is bounded by* 0.0352 *bit. If we xor 50 i.i.d. secret positions (as we do later in Section 6) and try to recover the new random bit from one oracle call, then we can instead obtain* 0.6255 *bit of information. Thus, from an information-theoretical perspective, the new framework is much more advantageous.*

3.2 Code generation

It has been demonstrated in previous research [RU08] that random sparse linear codes exhibit superior decoding performance and specific classes of Low-Density Parity-Check (LDPC) codes, such as [RSU01], can attain error-correction capabilities that approach the Shannon capacity. In this work, we present a straightforward code construction method that has shown remarkable results in our experiments.

We first borrow the concept of distance spectrum from [GJS16].

Definition 3 (Distance Spectrum [GJS16]). *For a binary vector* $\mathbf{h} \in \mathbb{F}_2^{n_0}$, *we define its distance spectrum* $D(\mathbf{h})$ *as*

$$D(\mathbf{h}) = \{d : 1 \leq d \leq \lfloor n_0/2 \rfloor, \textit{d classified as existing in } \mathbf{h}\},$$

where "existing in $\mathbf{h}$*" means there are two ones in* $\mathbf{h}$ *with distance d or* $(n_0 - d)$ *inbetween. A distance d can appear many times in the distance spectrum of a given bit pattern* $\mathbf{h}$. *We call this number the multiplicity of d.*

In our new attack, we first generate QC-LDPC codes with mb blocks of the parity-check matrix

$$\mathbf{H}_{\mathsf{ini}} = \begin{bmatrix} \mathbf{H}_{11} & \cdots & \mathbf{H}_{1b} \\ \vdots & \ddots & \vdots \\ \mathbf{H}_{m1} & \cdots & \mathbf{H}_{mb} \end{bmatrix},$$

where $\mathbf{H}_{ij}$ is the circulant matrix (or the negacyclic matrix in the q-ary case) generated from a binary vector $\mathbf{h}_{ij}$ for $1 \leq i \leq m, 1 \leq j \leq b$ with a low Hamming weight. We generate the vectors $\mathbf{h}_{ij}$ randomly with the constraint that only distances of multiplicity 1 are allowed in its distance spectrum. This can be done with high probability since the constructed LDPC codes are sparse. The key point in the design is that a length-4 cycle occurs in the associated Tanner graph if the multiplicity of a distance in the distance spectrum is larger than 2. By avoiding such patterns in a block, we can avoid many length-4 cycles; such attempts can improve the decoding performance as length-4 cycles can substantially hurt the decoding performance.

We select r rows of $\mathbf{H}_{\mathsf{ini}}$ (randomly or according to certain rules) to form a sub-matrix $\mathbf{H}'$ and append $-\mathbf{I}_{r\times r}$, where $\mathbf{I}_{r\times r}$ is the identity matrix. Thus, the parity-check matrix of the final generated code is

$$\mathbf{H} = \left[\mathbf{H}'_{r\times n_0 b} | -\mathbf{I}_{r\times r}\right] \tag{4}$$

Concatenated code construction. The LDPC codes generated from the above simple approach can serve as the outer code in the concatenated construction. The inner code can be any linear code such as a repetition code, the extended Hamming codes, and a further concatenation of the extended Hamming codes and repetition codes in [Ngo+21]. Moreover, we can include a soft-input-soft-output decoder (e.g., in [JZ98]) to utilize the soft-information. Note that in the soft-decoding procedure (e.g., the BP algorithm) of the outer code, only a distribution of each secret coefficient random variable is required; we could thus employ a code with an efficient maximum likelihood decoding procedure as the inner code allowing an efficient calculation of the soft output of the coefficient distribution.

In summary, such concatenated code construction enhances decoding capability and also balances decoding complexity, as the decoding of both outer and inner codes is efficient. This construction is particularly effective for lattice-based proposals or when the side-channel oracle exhibits a low level of accuracy.

4 Application to Kyber

In this section, we outline the details of how the new SCA-LDPC framework can be applied to Kyber. The attack is more effective for Kyber if we have side-channel leakages for both s_i and c_j. We demonstrate how to obtain these leakages, construct inner codes for them, and apply the outer LDPC decoder.

4.1 Basic key recovery attack

In the following, we explain the basic attack to obtain side-channel information about secret coefficients s_i. We focus on Kyber-768 mostly because the new protected implementation [Hei+22] that we target supports only this set of parameters. For Kyber-768, the secret key is $\mathbf{s} = (\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2)$, a ciphertext is a pair $(\mathbf{u}', \mathbf{v}')$. To decrypt a ciphertext, one computes $\mathbf{m} = \mathbf{Comp}_q(\mathbf{v} - \mathbf{s}^{\mathrm{T}}\mathbf{u}, 1)$, where $\mathbf{u} = \mathbf{Decomp}_q(\mathbf{u}', d_u) = (\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2)$, $\mathbf{v} = \mathbf{Decomp}_q(\mathbf{v}', d_v)$. The common practice [Rav+20] is to choose a ciphertext that leads to $\mathbf{m} = (0, 0, \ldots, 0)$ or $\mathbf{m} = (1, 0, \ldots, 0)$. In other words, all bits of the message are fixed to 0 except the first one. This can be done, for example, by setting $\mathbf{u}_0 = (k_u, 0, \ldots, 0)$,

$\mathbf{u}_1 = \mathbf{u}_2 = \mathbf{0}$ and $\mathbf{v} = (k_v, 0, \ldots, 0)$, where k_u, k_v are some numbers modulo q. In this case, the message bits are subject to the following equation.

$$\mathbf{m}[i] = \begin{cases} \mathbf{Comp}_q(k_v - k_u \cdot \mathbf{s}_0[0], 1), & i = 0 \\ \mathbf{Comp}_q(k_u \cdot \mathbf{s}_0[i], 1), & i \geq 1 \end{cases} \tag{5}$$

By choosing appropriate values for k_u and k_v, it is possible to force $\mathbf{m}[i]$ to always be zero for $i \geq 1$, while the value of $\mathbf{m}[0]$ depends on the first secret coefficient $\mathbf{s}_0[0]$. Since secret coefficients for Kyber-768 are taken from the range $[-2, \ldots, 2]$, some of the coefficients are encoded as 0, while others are encoded as 1. We can use several such ciphertexts with (possibly) different k_v and/or k_u to get an inner code of longer length. This way, using an oracle that distinguishes message $(1, 0, \ldots, 0)$ from $(0, 0, \ldots, 0)$, the attacker can get the distribution of a secret coefficient closer to the real value the more ciphertexts he uses.

There are restrictions for the values k_u and k_v: (1) these values are taken from the image of $\mathbf{Decomp}_q$; (2) k_u is chosen in a way such that $\mathbf{Comp}_q(k_u \cdot s, 1) = 0$ for any secret coefficient s (follows from Eq. (5)). Thus, one cannot use any code; even though it is possible to encode each secret coefficient with only $\lceil \log_2(5) \rceil = 3$ bits, for any fixed in-advance combination of 3 ciphertexts one cannot fully determine an arbitrary secret coefficient even with perfect oracle.

One way to solve this problem [She+22] is to choose ciphertexts adaptively based on the output of the oracle, but we take a different approach. Consider an FD oracle based attack, i.e., assume that we have a set of oracles $(\mathcal{O}_i)_{i \in (0..n-1)}$, where n is the length of the message. Given a ciphertext, the oracle $\mathcal{O}_i$ says if $\mathbf{m}[i] = 1$ or not. Essentially, the attacker calls all of these oracles at once, giving them the same ciphertext, this way he can get information about the whole message to be decrypted, the scenario is the same as in [Ngo+21]. The attacker can create a ciphertext in the following way, set $\mathbf{u}_0 = (k_u, 0, \ldots, 0)$, $\mathbf{u}_1 = \mathbf{u}_2 = \mathbf{0}$, and $\mathbf{v} = (k_v, k_v, \ldots, k_v)$, then

$$\mathbf{m}[i] = \mathbf{Comp}_q(k_v - k_u \cdot \mathbf{s}_0[i], 1), \tag{6}$$

i.e., i^{th} bit of the message depends on $\mathbf{s}_0[i]$. Thus, from one ciphertext the information about the block of 256 coefficients $\mathbf{s}_0$ can be obtained. Since there is no more restriction on $\mathbf{m}[i] = 0$ for $i \geq 1$, the amount of possible inner codes increases greatly. Table 3 shows an inner code from three ciphertexts built from (k'_u, k'_v) pairs, this code can be used to fully determine 256 secret coefficients with perfect oracles. Note that to create an actual ciphertext $(\mathbf{u}', \mathbf{v}')$ we need a pair (k'_u, k'_v) that maps to (k_u, k_v) with coefficient-wise function $\mathbf{Decomp}_q$. The next block of secret coefficients $\mathbf{s}_1$ can be retrieved by setting $\mathbf{u}_0 = \mathbf{0}$, $\mathbf{u}_1 = (k_u, 0, \ldots, 0)$, $\mathbf{u}_2 = \mathbf{0}$ and so on. Note that the attacker could choose different values in $\mathbf{v}$, this way different encodings can be used for different message bits (although all those encodings should have the same k_u) and this potentially opens up the possibility

Table 3: Example of an inner code for the secret coefficients. Each value from the range $[-2, \dots, 2]$ is encoded with 3 bits (columns of the table), therefore, the secret coefficient could be fully determined with just 3 oracle calls given that the oracle is perfect.

(k'_u, k'_v)	Secret coefficient				
	-2	-1	0	1	2
$(630, 14)$	0	1	0	1	1
$(706, 6)$	0	0	1	1	0
$(706, 10)$	0	1	1	0	0

of the adaptive attack. However, such an attack is more complicated since the set of allowed encodings given the fixed k_u is quite limited, and the attacker has to choose the same k_u for all 256 coefficients. We leave it as a potential follow-up work and focus on the situation where for all message bits there is a fixed in-advance encoding to be used.

The common approach in the literature is to use just an inner code for secret coefficients (without outer code) that makes the probability of getting the wrong coefficient to be very small (with real imperfect oracles), such that the probability to get all secret coefficients correctly is close to 1. In our approach, however, we use a shorter inner code that is not sufficient by itself, for example in our real attack from Section 6.1 we encode each secret coefficient with only 2 bits and encode the values -2 and 2 the same way, i.e., with only inner code it is impossible to differentiate between these values.

How to choose inner code. For the fixed in-advance code length ℓ we want to create an inner code C_ℓ that maximizes the information we get from the oracles with accuracy ρ. We solve this problem by considering the entropy of secret coefficients. Initially, each of them is distributed according to B_μ, whose entropy is $H(\mathsf{B}_\mu) \approx 2.03$, for $\mu = 2$. Each value $s \in \mathsf{B}_\mu$ is encoded as $C_\ell(s)$ – a binary string of length ℓ. Given an output string $\mathbf{y}$ of length ℓ from an oracle (note that $\mathbf{y}$ can be different from every $C_\ell(s)$, $s \in \mathsf{B}_\mu$), consider the probability $\mathbf{Pr}\left[\mathsf{B}_\mu = s \mid \mathbf{y}\right]$ for each $s \in \mathsf{B}_\mu$. As an example from Table 3, $\mathbf{Pr}\left[\mathsf{B}_\mu = 0 \mid 011\right] = 1$ for the perfect oracle, but it is less than 1 for an oracle with $\rho < 1$ since we could have reached this $\mathbf{y}$ from another coefficient.

To avoid ambiguity, we denote $\mathbf{y}_\rho$ as the output of the oracle with the accuracy ρ. The conditional distribution $\mathsf{B}_\mu|\mathbf{y}_\rho$ can be naturally defined as $\mathbf{Pr}\left[(\mathsf{B}_\mu|\mathbf{y}_\rho) = s\right] = \mathbf{Pr}\left[\mathsf{B}_\mu = s|\mathbf{y}_\rho\right]$. Now, the difference between the entropy values $H(\mathsf{B}_\mu) - H(\mathsf{B}_\mu|\mathbf{y}_\rho)$ shows how much information the output $\mathbf{y}_\rho$ gives. To assess how good the code is, we can compute the expectation of this information as

$$I(C_\ell) = \sum_{\mathbf{y}_\rho \in \{0,1\}^\ell} (H(\mathsf{B}_\mu) - H(\mathsf{B}_\mu | \mathbf{y}_\rho)) \cdot \mathbf{Pr}\left[Y = \mathbf{y}_\rho\right],$$

where Y is a random variable that describes the output of an oracle with accuracy ρ on a random secret coefficient. The probability of the specific oracle's output is computed as follows.

$$\mathbf{Pr}\left[Y = \mathbf{y}_\rho\right] = \sum_{x \in \mathrm{supp}(\mathsf{B}_\mu)} \rho^{d(\mathbf{y}_\rho, C_\ell(x))} (1-\rho)^{\ell - d(\mathbf{y}_\rho, C_\ell(x))} \mathbf{Pr}\left[\mathsf{B}_\mu = x\right],$$

where $d(\cdot,\cdot)$ is the Hamming distance. To decode a received word $\mathbf{y}_\rho$, one computes conditional probability $\mathsf{B}_\mu | \mathbf{y}_\rho$ of secret coefficient, i.e. we use maximum-likelihood decoding approach.

4.2 Improving the attack using LDPC

The basic attack allows us to compute the conditional distribution for each secret coefficient using the inner code. Now, following our framework, we create an outer LDPC code. For it to work, we also need a way to get information about parity checks c_i. Let us describe how to create a ciphertext corresponding to a parity check. Consider an example: Let $\mathbf{u}_1, \mathbf{u}_2$ and $\mathbf{v}$ be as above, but $\mathbf{u}_0 = k_u + k_u x^2$, then

$$\mathbf{s}^{\mathrm{T}}\mathbf{u} = k_u\left((\mathbf{s}_0[0] - \mathbf{s}_0[n-2]) + (\mathbf{s}_0[1] - \mathbf{s}_0[n-1])x + (\mathbf{s}_0[2] + \mathbf{s}_0[0])x^2 + \ldots\right).$$

Looking at the first message bit

$$\mathbf{m}[0] = \mathbf{Comp}_q(k_v - k_u \cdot (\mathbf{s}_0[0] - \mathbf{s}_0[n-2]), 1)$$

and comparing it to Eq. (6), one can recover $c_0 \leftarrow \mathbf{s}_0[0] - \mathbf{s}_0[n-2]$ using a similar approach as in recovering $\mathbf{s}_0[0]$ with $\mathcal{O}_0$. However, c_0 lies in the range $[-4, \ldots, 4]$, therefore the recovery process is more complicated. However, we still use several different ciphertexts to get an inner code for the check variables. In other words, there are two inner codes: one for secret coefficients, and another one for check variables. Each of them helps us to compute conditional distributions, which we use with outer LDPC code.

Now, let us represent c_0 as a vector $\mathbf{h}_0$ with values from $\{-1, 0, 1\}$ such that $c_0 = \mathbf{h}_0^{\mathrm{T}}(\mathbf{s}_0[0], \ldots, \mathbf{s}_0[n-1])$. In general, if $\mathbf{u}_0 = k_u \cdot \sum_{j=1}^{w} x^{i_j}$, then $\mathbf{h}_0$ is a vector with w nonzero entries at the positions $(-i_j) \bmod n$, where the entry is 1 if and only if $i_j = 0$. Let $\mathbf{H}_0$ be a negacyclic matrix of the vector $\mathbf{h}_0$. With this ciphertext, the i^{th} message bit is connected to the i^{th} row of $\mathbf{H}_0(\mathbf{s}_0[0], \ldots, \mathbf{s}_0[n-1])^{\mathrm{T}}$. Note that, unlike in Section 3.1, c_i is the sum of secret coefficients, possibly multiplied by -1. However, this does not significantly affect the result since from the distribution of the coefficient it is trivial to obtain the distribution of the negative coefficient and vice versa. Thus, we still call c_i the sum of secret coefficients.

Let $\mathbf{u}_r = k_u \cdot \sum_{j=1}^{w} x^{i_j^{(r)}}$, $r \in \{0, 1, 2\}$. Ciphertext $(\mathbf{u}, \mathbf{v})$ with the help of oracles $\mathcal{O}_i$ reveals information about 256 parity checks. The parity-check matrix of the outer LDPC code in this case is of the form

$$\mathbf{H}_{\mathsf{ini}} = \left[\mathbf{H}_0|\mathbf{H}_1|\mathbf{H}_2\right],$$

where $\mathbf{H}_j$ is the negacyclic matrix obtained from the vector connecting c_0 and $\mathbf{s}_j$. Due to the FD oracle, parity checks $c_1, \ldots, c_{n-1}$ must be negacyclic shifts of c_0. We only demonstrated the parity-check matrix for the outer LDPC code consisting of block of 256 checks, but there could be several such blocks. Note that in general, the polynomials $\mathbf{u}_r$ do not have to use the same w.

There are three main ways to increase the success probability of the attack.

1. Increase the length of the inner code for the secret coefficients. Querying oracles as in Section 4.1 leads to a more accurate distribution for each coefficient.
2. Similarly, increase the length of the inner code for the check variables, i.e., fix the indexes $i_j^{(r)}$ and use different (k_u, k_v).
3. Increase the number of check blocks. The resulting parity-check matrix of the LDPC code $\mathbf{H}_{\mathsf{ini}}$ consists of $3 \times m$ blocks of negacyclic matrices, where m is the number of "unique" parity checks $c_0, c_n, c_{2n}, \ldots$

Creating the best inner code for the check variables that maximizes the amount of information is a challenging task. An educated guess would be the most accurate way to describe our approach to tackling this problem.

5 Application to HQC

In this section, we describe the detailed attack on HQC. $\mathcal{O}_{\text{HQC}}$ denotes a general side-channel-based PC oracle for HQC, referenced prior-art assumes timing leakage, but this is not required. We treat a key-misuse oracle as a chosen-ciphertext side-channel oracle with 100% oracle accuracy.

5.1 Key-recovery attack with $\mathcal{O}_{\text{HQC}}$

In [Guo+22] the authors presented a plaintext checking (PC) oracle based on timing information due to the use of rejection sampling. In this section, we describe how the PC attack works and then explain how we can improve it by using our new SCA-LDPC framework, which is based on coding theory.

Currently, HQC makes use of so-called rejection sampling in the CPA secure encryption function [Agu+20; Guo+22]. The rejection sampling algorithm is used to construct random vectors with a specific Hamming weight ω. It works by random sampling of bit positions in the vector, and if some positions are sampled twice, they are rejected. Straight-forwardly implemented, this algorithm leaks timing information due to the inherently random number of rejections that occur. The HQC implementations tested in [Guo+22] leak timing information mainly through the use of so-called "seedexpander" calls. The output of the seedexpander function is deterministic pseudo-randomness given by an eXtendable Output Function (XOF). The rejection sampling algorithm uses the seedexpander function to generate relatively large blocks of randomness, at a time. The timing distribution, therefore, is highly dependent on the number of seedexpander calls needed. The minimum number of seedexpander calls occurs when there are no rejections in the rejection sampling algorithm. In practice, we classify timing measurements based on the number of *additional* seedexpander calls. They are each related to one of the four[2] distributions $\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$, listed in increasing order of rarity.

Prior to the publication of the referenced work, it was believed that this randomness was only dependent on values known to the attacker, in this case, the plaintext $\mathbf{m}$. The assumption then was that constant time implementation was not needed for the rejection sampling algorithm. Certainly, it was shown in [Guo+22] that this assumption is problematic. Although $\mathbf{m}$ is indeed known to the attacker, the result of the implicitly carried out comparison $\mathbf{m}' \overset{?}{=} \mathbf{m}$ is not. Here $\mathbf{m}' = \text{decode}(\mathbf{c} + \mathbf{e}')$ and $\mathbf{e}'$ is a extra noise supplied by the attacker.

The authors showed a key-recovery attack where, by using the timing information due to rejection sampling, knowledge of $\mathbf{m}' \overset{?}{=} \mathbf{m}$ is leaked. The attack required 866,000 so-called "idealized oracle" ($\mathcal{O}_{\text{HQC}}^{\text{ideal}}$) queries for the 128-bit security setting. The idealized oracle assumes a noise-free environment where a single timing measurement is sufficient to determine the membership of $\mathcal{S}_j$ (where $j = 3$ in [Guo+22]). Unfortunately, this is not sufficient for a 100% correct oracle, due to reasons explained in the following paragraph.

What follows is a high-level summary of the referenced attack; A plaintext $\mathbf{m}$ is selected according to some criteria useful for the distinguisher. In the case of timing leakage, the distinguishing property is such that the selected $\mathbf{m}$ results in the timing distribution $\mathcal{S}_3$, since it is the one most easily distinguished. The probability of for any random $\mathbf{m}'$, where $\mathbf{m}' \neq \mathbf{m}$, resulting in the same $\mathcal{S}_3$ timing distribution is low (0.58% per [Guo+22]). In other words, $\mathbf{m}' \overset{?}{=} \mathbf{m}$ can be distinguished with a high, yet-not-complete, advantage.

[2] Strictly, there is no upper bound, but the practical benefit of finding a value for $\mathcal{S}_{\geq 4}$ is not worth the exponential effort required [Guo+22].

A ciphertext $\mathbf{c}' = (\mathbf{u}, \mathbf{v})$ is crafted in the next step such that $\mathbf{r}_1$ is $\mathbf{1} \in \mathcal{R}$ and $\mathbf{r}_2$ and $\mathbf{e}$ is $\mathbf{0} \in \mathcal{R}$. By Eq. (3) this results in

$$\mathbf{v} - \mathbf{u} \cdot \mathbf{y} = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e} - (\mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2) \cdot \mathbf{y} = \mathbf{mG} - \mathbf{r}_1 \cdot \mathbf{y} = \mathbf{mG} - \mathbf{y} \quad (7)$$

which makes $\mathbf{y}$ the only remaining error for the decoder to correct. Note too that by knowledge of $-\mathbf{y} = \mathbf{y}$ it is a simple computation to find the rest of the private key, since $\mathbf{x} = \mathbf{s} - \mathbf{h} \cdot \mathbf{y}$. Calculating $\mathbf{x}$ is quite unnecessary, however, since it is not used in decapsulation.

Plainly, this crafted ciphertext is invalid and will be rejected in the ciphertext comparison step of the decapsulation. However, a valid ciphertext is not required due to the timing leakage in the XOF via the non-constant time rejection sampling algorithm. The reencryption step immediately preceding the comparison derives the values of $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{e}$ from the XOF seeded by $\mathbf{m}$. The single bit information $\mathbf{m}' \overset{?}{=} \mathbf{m}$ leaks prior to the ciphertext comparison step.

Hall et al. proposed in [HGS99] a way to recover $\mathbf{y}$; An additional error vector $\mathbf{e}'$ is added to $\mathbf{c}'$. $\mathbf{e}'$ is of just sufficient weight to cause a decoding failure (i.e. $\mathbf{m}' \neq \mathbf{m}$ is leaked). The basic attack then simply iterates through each bit $0 \leq i \leq N$ of $\mathbf{e}'$ not already flipped to find those positions that if flipped would result in a decoding success. If this is the case for any value of i this indicates that the bit was already flipped in $\mathbf{y}$ in the ciphertext.

However, this technique alone is not sufficient to provide decisions on all bits in the ciphertext. The reason is twofold. First, unflipping a bit in the error pattern given to the RMRS decoder does not guarantee a decoding success, and secondly due to the possibility that both $\mathbf{m}'$ and $\mathbf{m}$ result in the timing distribution $\mathcal{S}_3$, even though $\mathbf{m}' \neq \mathbf{m}$. This is modeled by $\mathcal{O}^{\text{ideal}}_{\text{HQC}}$, the idealized oracle from [Guo+22], which though noise-free, is not 100% correct.

The first problem is solved by using many different error patterns $\mathbf{e}'$. The second was solved by majority voting, i.e. by gathering three or more decisions for every bit. Both of these solutions drive up the number of required oracle calls, even in the ideal timing leakage setting. For the 128-bit security level, this number adds up to 866,000 oracle calls [Guo+22].

5.2 New improved attack using LDPC codes.

What follows is a description of the new attack listed in Algorithm 1, a PC oracle $\mathcal{O}_{\text{HQC}}$ is assumed. Like in the original attack [Guo+22] we select a plaintext with good side-channel detection properties (in the original case this is a timing property).

The next step is to construct a $N \times N$ regular cyclic LDPC parity-check matrix $\mathbf{H}_{\text{ini}}$ without cycles of length 4, with a good decoding performance. $\mathbf{H}_{\text{ini}}$ has a row-weight of W. This construction is detailed in Section 3.2, with $(m = 1, b = 1)$. The first row of $\mathbf{H}_{\text{ini}}$ is the vector $\mathbf{h}_{\text{ini}}$.

Algorithm 1: $\mathcal{O}_{\mathrm{HQC}}^{0\text{=repeat}}$ denotes a PC oracle which is repeated as necessary (determined by empirical study) to achieve better than nominal error rate in the case of decoding failure; decoding successes are never repeated. $\mathcal{O}_{\mathrm{HQC}}^{1\text{=repeat}}$ works in a similar but opposite fashion.

Algorithm 1 HQC new attack algorithm

```
Input: $\mathcal{O}_{\mathrm{HQC}}$, public key
Output: y
Select plaintext $\mathbf{m}$                          ▷ With good side channel distinguishing properties
Generate sparse vector $\mathbf{h}_{\mathrm{ini}}$                      ▷ According to Section 3.2
Construct $\mathbf{H}_{\mathrm{ini}}$                                ▷ From $\mathbf{h}_{\mathrm{ini}}$ by cyclic shifts
Craft $\mathbf{c}'$ with $\mathbf{r}_2 = \mathbf{0}, \mathbf{e} = \mathbf{0}$ and $\mathbf{r}_1 = \mathbf{h}_{\mathrm{ini}}$
$\mu \leftarrow 0^N$                                       ▷ Initialize message
loop
    $\mathbf{e}' \leftarrow 0^N$,
    $\mathcal{B}' \leftarrow$ random subset of size $(d_{RS} - 1)/2$ from $\{0, \ldots, n_1 - 1\}$
    for all $B' \in \mathcal{B}'$ do                        ▷ Flip $(d_{RS} - 1)/2$ RM-blocks
        Flip block $B'$ in $\mathbf{e}'$
    end for
    $B \xleftarrow{\$} \{0, \ldots, n_1\} \setminus \mathcal{B}'$               ▷ Select a random unflipped block
    $\mathcal{I}^B \leftarrow \{Bn_2, \ldots, B(n_2 + 1) - 1\}$
    $\mathcal{I}^B_{\mathbf{e}'}, \mathcal{I}^B_0, \mathcal{I}^B_1, \leftarrow \emptyset, \emptyset, \emptyset,$
    while $\mathcal{O}_{\mathrm{HQC}}^{0\text{=repeat}}(\mathbf{c}' + \mathbf{e}')$ do       ▷ Find an initial error pattern for block $B$
        $\mathcal{I}^B_{\mathbf{e}'} \leftarrow \mathcal{I}^B_{\mathbf{e}'} \cup \{i\}$, where $i \xleftarrow{\$} \mathcal{I}^B$
        $\mathbf{e}'[i] \leftarrow 1$
    end while
    for all $i \in \mathcal{I}^B_{\mathbf{e}'}$ do                        ▷ Minimize the error pattern
        $\mathbf{e}'[i] \leftarrow 0$                                ▷ Unflip bit in error pattern
        if $\mathcal{O}_{\mathrm{HQC}}^{1\text{=repeat}}(\mathbf{c}' + \mathbf{e}')$ then
            $\mathcal{I}^B_0 \leftarrow \mathcal{I}^B_0 \cup \{i\}$                   ▷ Satisfied parity check, add $i$ to $\mathcal{I}^B_0$
            $\mathbf{e}'[i] \leftarrow 1$                            ▷ Restore bit in error pattern
        end if
    end for
    for all $i \in \left(\mathcal{I}^B \setminus \mathcal{I}^B_{\mathbf{e}'}\right)$ do                  ▷ Find unsatisfied parity checks
        if $\mathcal{O}_{\mathrm{HQC}}(\mathbf{c}' + \mathbf{e}')$ then
            $\mathcal{I}^B_1 \leftarrow \mathcal{I}^B_1 \cup \{i\}$                        ▷ If found, store in $\mathcal{I}^B_1$
        end if
    end for
    Select rows $i \in (\mathcal{I}^B_0 \cup \mathcal{I}^B_1)$ from $\mathbf{H}_{\mathrm{ini}}$ and add to $\mathbf{H}'$
    Construct $\mathbf{H} = [\mathbf{H}' | \mathbf{I}]$
    $\mu \leftarrow \mu \mid 0^{\#\{\mathcal{I}^B_0\}} \mid 1^{\#\{\mathcal{I}^B_1\}}$
    $\mathbf{y} \leftarrow \mathrm{Decode}_{\mathbf{H}}(\mu)[0..n]$                 ▷ Decoder returns the error vector
    if $\mathbf{y}$ correct then                             ▷ Try to decrypt a valid message
        return $\mathbf{y}$
    end if
end loop
```

We craft a special ciphertext $\mathbf{c}'$ where $\mathbf{r}_2 = \mathbf{0}, \mathbf{e} = \mathbf{0}$ and $\mathbf{r}_1 = \mathbf{h}_{\text{ini}}$. Similarly to the case given by Eq. (7) above, this results in

$$\mathbf{v} - \mathbf{u} \cdot \mathbf{y} = \ldots = \mathbf{mG} - \mathbf{r}_1 \cdot \mathbf{y} = \mathbf{mG} - \mathbf{h}_{\text{ini}}\mathbf{y} \tag{8}$$

which makes the added noise that the decoder has to correct equal to $\mathbf{h}_{\text{ini}}\mathbf{y}$. In other words, each bit position i in $\mathbf{c}'$ correspond to the result of a parity-check equation over $\mathbf{y}$, given by $\mathbf{h}_{\text{ini}} >> i$ (cyclic shift by i steps) due to the cyclic nature of our LDPC code.

The Reed-Muller (RM) and Reed-Solomon (RS) concatenated (RMRS) decoder, used in HQC, can be attacked in two stages. First we select $(d_{RS} - 1)/2$ outer RM blocks (each RM block decodes to one RS symbol) to flip in $\mathbf{c}'$ (by XOR with $\mathbf{e}'$). This results in a state where if one more block is flipped it will result in a decoding error in the RS decoder. A decoding failure such as that would be detected by $\mathcal{O}_{\text{HQC}}$. We randomly select another block which we denote B.

The next stage is to find which bits $\mathcal{I}^B_{\mathbf{e}'}$ to flip in the block B that results in a decoding failure. We do this by flipping bits $i \in \mathcal{I}^B_{\mathbf{e}'}$ such that $\mathbf{e}'[i] = 1$ in block B until a RM decoding failure occurs. This propagates as a failure symbol to the RS decoder which is already on the brink of being overwhelmed. This results in a state where $\mathbf{c}' + \mathbf{e}'$ fails to decode due to too much additional noise in the block B partition of $\mathbf{e}'$.

An aside on oracle accuracy. The LDPC code helps with recovery from bad oracle decisions. However, the stateful nature of the new algorithm can cause certain poor oracle decisions to propagate and result in the algorithm ending up in a bad state. Such errors occur naturally more often for less accurate oracles. We compensate for these effects by introducing extra confirmation calls to those oracle decisions which are most sensitive. These are denoted in Algorithm 1 by $\mathcal{O}^{r\text{=repeat}}_{\text{HQC}}$, where $r \in \{0, 1\}$ indicates which Oracle outputs are repeated for confirmation. $\mathcal{O}^{0\text{=repeat}}_{\text{HQC}}$ means decoding failures are confirmed but decoding successes are not. The number of repeated oracle calls is determined by empirical study.

After finding an error pattern resulting in decoding failure, the next step is to reduce the number of flipped bits, in block B. The goal is to find the minimal pattern that still results in a decoding failure. We do this by unflipping each of the flipped bits $i \in \mathcal{I}^B_{\mathbf{e}'}$ in block B. This results in one of two cases:

1. If we get a decoding success we record it in $\mathcal{I}^B_0$ for later use, undo the flip and then move on to select another bit $i \in \mathcal{I}^B_{\mathbf{e}'}$.
2. If we still get a decoding failure we try again with another flipped bit $i \in \mathcal{I}^B_{\mathbf{e}'}$.

Once we have run out of flipped bits in $\mathcal{I}^B_{\mathbf{e}'}$ to check, we have achieved a minimal bit pattern in $\mathcal{I}^B_0$ for block B that results in decoding failure. That is, the set $\mathcal{I}^B_0$ contains those bits that result in a decoding success if any are unflipped.

Conversely, when flipped, they have been unambiguously shown to increase the noise for the RMRS decoder. All bits in $\mathcal{I}_0^B$ can therefore reliably be assumed to correspond to a satisfied parity check. So, for each bit $i \in \mathcal{I}_0^B$ we construct[3] our sub matrix $\mathbf{H}'$ by the selection of row i of $\mathbf{H}_{\text{ini}}$.

Working from the minimal decoding failure pattern ($\mathbf{e}'[i] = 1 \,\forall\, i \in \mathcal{I}_0^B$ and $\mathbf{e}'[i] = 0 \,\forall\, i \notin \mathcal{I}_0^B$) for RM block B we can now flip bits that so far have been left untouched ($i \notin \mathcal{I}_{\mathbf{e}'}^B$), one at a time. For each flip, if it results in a decoding success, then we record it in $\mathcal{I}_1^B$. Such a bit must mean that by flipping it we reduce the noise that the RMRS decoder has to handle. Therefore, this bit can be reliably assumed to correspond to an unsatisfied parity-check equation, or a '1' in the vector $\mathbf{h}_{\text{ini}}\,\mathbf{y}$. When all bits have been tested we extend our sub matrix $\mathbf{H}'$ by the selection of all rows $i \in \mathcal{I}_1^B$ of $\mathbf{H}_{\text{ini}}$.

At this time in the algorithm, r number of parity-check equations have been collected in $\mathbf{H}'$. The remaining step is to construct parity-check matrix $\mathbf{H} = \left[\mathbf{H}'_{r\times n}|\mathbf{I}_{r\times r}\right]$ and a message vector

$$\mu = \left[\mathbf{0}^n|\mathbf{0}^{\#\{\mathcal{I}_0^{B^0}\}}|\mathbf{1}^{\#\{\mathcal{I}_1^{B^0}\}}\ldots|\mathbf{0}^{\#\{\mathcal{I}_0^{B^t}\}}|\mathbf{1}^{\#\{\mathcal{I}_1^{B^t}\}}\right] \tag{9}$$

in such a way that we have n zeroes, each representing an unknown bit-value of $\mathbf{y}$ to be recovered. The message is appended by the following redundancies: a single 0 for each satisfied parity-check equation hitherto selected ($i \in \mathcal{I}_0^B$) and a 1 for each unsatisfied parity check ($i \in \mathcal{I}_1^B$). We do this for all t blocks B that have so far been selected.

We try to decode the message μ and recover $\mathbf{y}$ from the first n bits. We use $\mathbf{H}$ as input and a suitable decoder such as sum-product or the min-sum approximation.

If the decoding is not successful we unflip all bits in block B and unflip all other blocks. Then we restart the algorithm (using the same ciphertext) and select another block. The old $\mathcal{I}_0^B$ and $\mathcal{I}_1^B$ are saved and re-used in the next decoding attempt. We continue until successful.

In some cases (for less accurate oracles) one might still fail to decode even after all outer RM blocks have been exhausted. In such cases, one can simply save μ and $\mathbf{H}'$ and continue extending them by restarting the algorithm.

6 Experiments

In this section, we show the results of simulations and real-world experiments for Kyber and HQC.

[3] or extend if this is not the first selected block/iteration of the algorithm

6.1 Masked Kyber

Software simulations

We introduce software simulations, where we fix the accuracy ρ of each oracle $\mathcal{O}_i$ to be the same.

The attack improves as the weight of the rows in the parity matrix increases. However, the decoding time increases exponentially with it. In the course of experiments, we found that the value $w = 2$ works best, i.e., the parity-check matrix consists of negacyclic matrices with row weight 2. For Kyber-768, this means that each check variable is the sum of 6 secret coefficients.

The three main parameters of the attack are m_0, m_1 and m_2, where m_0 and m_2 are the lengths of the inner code for the secret coefficients and the check variables, resp., m_1 is the number of blocks of check variables. Recall that Kyber-768 has 3 blocks of 256 secret coefficients, and we assume that from one power trace we get information about all 256 message bits. This means that we need $3m_0$ and $m_1 \cdot m_2$ traces to get the distributions for secret coefficients and check variables, respectively. The interested reader is referred to Tables 8 and 9 for the actual codes used in the simulation and in the real attack.

We evaluate our methodology against the majority voting technique, a conceptually simple coding approach that can be considered as a repetition code. Majority voting is a typical approach to ensure that a single secret coefficient can be recovered with high accuracy. This approach has been selected as the baseline attack method due to its relevance as the most frequently used coding scheme for attacking Kyber in previous literature (e.g., in [She+22]). For majority voting, we choose the code as in Table 3 and use t votes, i.e., the actual code is repeated t times. We run 1000 tests and compute the average number of wrong secret coefficients, the attack is considered successful if this number is less than 1. For our approach, we choose m_0, m_1, and m_2 such that the total number of traces is minimized and the average number of errors is close to majority voting. We run 100 tests, and all tests are done with randomly generated secret keys. The results for a wide range of accuracy levels are shown in Table 4.

Real-world experiments

We conduct our experiments in the ChipWhisperer toolkit, including the ChipWhisperer-Lite board, the CW308 UFO board, and the CW308T-STM32F4 target board with a 32-bit ARM Cortex-M4 CPU. We target the mkm4 library[4] in [Hei+22] implementing a first-order masked version of Kyber. The library is compiled using the -O3 optimization level, which is typically harder to attack [She+22; Ngo+21]. The target board is run at 24 MHz, and the traces are sampled at 24 MHz.

[4] https://github.com/masked-kyber-m4/mkm4

Table 4: Comparison with the majority voting for full-key recovery. t is the number of votes cast, values in the brackets are m_0, m_1 and m_2, resp.

$\rho = 0.995$	Number of traces	Average number of errors
Majority Voting ($t = 3$)	27 (ref)	$0.21/768$
Our Method $(2, 1, 4)$	10 (−63%)	$0.37/768$
$\rho = 0.95$	Number of traces	Average number of errors
Majority Voting ($t = 7$)	63 (ref)	$0.47/768$
Our Method $(3, 4, 2)$	17 (−73%)	$0.16/768$
$\rho = 0.9$	Number of traces	Average number of errors
Majority Voting ($t = 11$)	99 (ref)	$0.67/768$
Our Method $(4, 3, 4)$	24 (−75.8%)	$0.46/768$

We attacked the function `masked_poly_tomsg` in the first draft of the work and it was the first power analysis attack on an open-source masked implementation of Kyber, as far as we know. Then we switched to the function `masked_poly_frommsg` similarly to [DNG22]. With this approach, real oracles from side-channel leakages have better accuracy, leading to a lower amount of traces.

The function `masked_poly_frommsg` (shown in Algorithm 2) maps each masked polynomial coefficient to a corresponding message bit during decapsulation. In one loop the function works on the message bits XORed with random bits; on the other loop it works with these random bits themselves. Obtaining a power trace for these two loops allows us to retrieve information about all message bits and implement the FD oracles $\mathcal{O}_i$.

The attack scenario is the same as in [Ngo+21]. First, there is the profiling stage during which, using the profiling device D_1, we collect 100,000 power traces of the function `masked_poly_frommsg`. It is done by generating a random message which is encrypted using the device's public key, the resulting ciphertext is passed to the measured by the ChipWhisperer decapsulation function. Each byte of the message is computed in the same way, and the power traces corresponding to each byte are similar. Thus, we can train only 8 neural network models, one for each bit of the byte. Models are trained for up to 100 epochs. The interested reader is referred to Table 10 for the architecture of the model.

Each of the 8 models simulates the 32 oracles $\mathcal{O}_{i+8j}$, $j = 0, \ldots, 31$, with some accuracy ρ_i. The oracle behaves like a binary symmetric channel with success probability ρ_i, but the model provides soft values, which can be treated as the probabilities of output being 1 or 0 from the model's perspective. Thus, the real-world attack is more powerful since there is more information we can work with.

After the profiling stage, there is the attacking stage. The assumption is that the attacker has access for a (relatively) short period of time to a similar device D_2. After collecting power traces for decapsulation on chosen ciphertexts, the

Algorithm 2 The attacked function in Kyber.CPAPKE.Dec() (from [Hei+22])

```
masked_poly_frommsg(uint16_t poly[2][256], uint8_t msg[2][32])

... /* initialization */
for $i = 0$ to $31$ do
    for $j = 0$ to $7$ do
        mask = -((msg[0][i] >> j) & 1)
        poly[0][8*i+j] += mask & ((KYBER_Q+1)/2)
    end for
end for
for $i = 0$ to $31$ do
    for $j = 0$ to $7$ do
        mask = -((msg[1][i] >> j) & 1)
        poly[1][8*i+j] += mask & ((KYBER_Q+1)/2)
    end for
end for
...
```

Table 5: Accuracy of recovering particular bit for models. Device D_1 is the profiling device, and D_2 is the device to be attacked.

Device	ρ_0	ρ_1	ρ_2	ρ_3	ρ_4	ρ_5	ρ_6	ρ_7
D_1	0.9651	0.9986	0.9985	0.9985	0.9992	0.9995	1.0000	1.0000
D_2	0.9390	0.9811	0.9923	0.9023	0.9654	0.8940	0.9404	0.9873

attacker's goal is to recover the key using the trained models. The Table 5 shows the accuracy ρ_i of recovering i^{th} bit for devices D_1 and D_2.

The experimental results (shown in Table 6) with the average oracle accuracy of 0.9502 are better than the simulation results with an accuracy of 0.95. There are two reasons for this: (1) real models provide soft values, making the attack more powerful; (2) In the simulation, the accuracy of each bit is the same, but for our LDPC approach, it is more beneficial for some bits to be more reliable than others.

Table 6: Real-world attack results on the first-order masked Kyber-768. We performed 100 runs of the attack with a random secret key for each run.

	Number of traces	Average number of errors
Majority Voting ($t = 11$)	99	$0.34/768$
Our Method $(2, 2, 3)$	12	$0.82/768$

On the other hand, the success of majority voting approach depends on the worst bit position. In other words, in the real world majority voting works worse since the bottleneck is the worst bit. The real attack with accuracy from Table 5 uses $t = 11$ votes, i.e. in total we need 99 traces (instead of 63 as in Table 4). In this case, our framework uses 86% fewer traces.

6.2 HQC

In order to test the new attack strategy against HQC it is advantageous to make as close to an apples-to-apples comparison as possible against the results of [Guo+22]. To this end, we model the PC oracle as follows; The success probability for an oracle query is determined by ρ_0 and ρ_1, which are the probabilities of correctly classifying decoding failures and decoding successes, respectively. For the case of the ideal HQC timing oracle used in [Guo+22] these values are listed in Table 7 and correspond to $\rho_0 = \rho_f$ and $\rho_1 = \rho_s$. We label the ideal oracle $\mathcal{O}_{\text{HQC}}^{\text{ideal}}$.

Table 7: Ideal HQC timing oracle, $\mathcal{O}_{\text{HQC}}^{\text{ideal}}$, as modelled with ρ_f and ρ_s.

Reported as / Real	decoding failure	decoding success
decoding failure	$\rho_f = 0.9942$	$1 - \rho_f = 0.0058$
decoding success	$1 - \rho_s = 0$	$\rho_s = 1.0$

Simulating real-world attacks with noisy measurements can be done by selecting other values of ρ_0 and ρ_1. For simplicity, we introduce ρ as a single representative value of PC oracle accuracy, where $\rho = \rho_0 = \rho_1$. We label the corresponding oracle $\mathcal{O}_{\text{HQC}}^{\rho}$.

By empirical study (see Fig. 7) we have selected a row weight of $W = 50$ in the constructed LDPC code (for hqc-128). This is close to the upper limit of our code generation algorithm. Using a bigger W would occasionally require a more advanced algorithm with backtracking of the random walk. Regardless, the decoding appears to suffer in reliability for values $W > 50$. Smaller values of W require more parity checks and thus make the attack slower.

Some interesting ρ values, corresponding to real attacks, are $\{1.0, 0.995, 0.95, 0.9\}$. In Fig. 2 we show the results of simulations using the various oracle models we have described so far. The results for $\mathcal{O}_{\text{HQC}}^{\text{ideal}}$ indicate an 86.6 times improvement over the original attack [Guo+22].

We have validated our attack by running a real timing oracle on a Ubuntu 20.04 LTS laptop with Intel Core i5-7200@2.50GHz. Measurement noise was reduced by turning of hyper-threading and by running in recovery mode. We used 2^{18} measurements to generate a profile, first of a decoding success and again of a decoding failure. Measuring 8 decapsulations resulted in an oracle accuracy of $\rho_{\mathcal{O}_{\text{HQC}}^{\text{real}}} = 0.951$ as determined by 1000 trials. The simulated results for $\mathcal{O}_{\text{HQC}}^{0.95}$

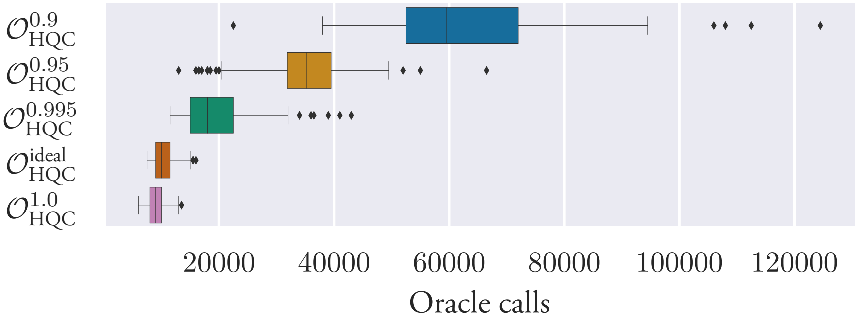


Figure 2: Experiment for hqc-128. The median number of oracle calls for successful key recovery, are 59500, 35250, 18000, 10000, and 9000 respectively for the listed oracles. For each oracle model, 100 key-recovery simulations ran to completion.

indicate a real-life key recovery attack of hqc-128 can be done by measuring $2^3 \times 35250 \approx 2^{18}$ decapsulation calls.

6.3 Discussions

In this section, we present discussions on the performance of the new SCA-LDPC framework, including its information-theoretical advantages and limitations. Furthermore, we compare the SCA-LDPC framework with the inner-symbol error correction method proposed in [Ngo+21] and highlight the advantages of the former.

A non-rigorous information theoretical bound. Assuming that a single side-channel measurement provides a certain amount of information (denoted by I bits), and considering the fact that there are k secret symbols that are independently generated from a distribution with entropy E bits, it is possible to calculate a lower bound for the number of measurements required. This can be accomplished by dividing $k \cdot \mathsf{E}$ by I. Estimation of I can be performed by considering each recovered message bit as a Bernoulli variable, with a specified probability ρ_i of being correct. It is noteworthy that the value of ρ_i may vary for different secret positions. This information-theoretical estimation is approximate in nature. It is subject to limitations arising from the simplicity of the Bernoulli model. Additionally, near-optimal source coding and channel coding are required to match the derived lower bound. Notwithstanding these limitations, the estimation suggests the possibility of improvement, though the extent of such improvement may be constrained.

The aforementioned lower bound is equivalent to the well-known Shannon source coding bound when the accuracy of the oracle is 100%, which can be used to characterize the source compression gain. The results obtained from the FD oracle based attack on the scheme Kyber-768 exactly meet the lower bound of 7 traces. Conversely, for the PC oracle based attack on hqc-128, 1324 parity checks were required, a factor of 2.1 times the lower bound of 628 checks.

It has been observed that the difference between the simulated results and the lower bound increases as the oracle accuracy decreases. For example, in the case of the PC oracle based attack on hqc-128, when the oracle accuracy drops to 0.95, the ratio of the simulated parity checks to the lower bound increases to approximately 2.7, as calculated by 2396/880. For the FD oracle based real-world attack on Kyber-768, based on the message recovery accuracy data presented in the second row of Table 5, the lower bound was determined to be 9, which is slightly lower than the 12 traces utilized in the actual attack.

Limitations. Despite the remarkable reduction of necessary side-channel measurements, a gap remains between actual performance and our non-rigorous information-theoretical lower bound. This gap may be attributed to the requirement of an extremely long codeword, potentially in the range of tens of millions of bits, for the LDPC codes to approach optimality. Additionally, it may be a result of the simplicity and inadequacy of our current code-construction method. More sophisticated LDPC code construction techniques could further reduce the required number of measurements.

Our method vs. inner-symbol error correction. The new SCA-LDPC framework utilizes a system of sparse parity checks to interconnect all the secret symbols. As a result, accurately determined symbols can be utilized to rectify incorrectly determined symbols, categorizing this method as inter-symbol error correction. On the other hand, the method presented in [Ngo+21] falls under the category of inner-symbol error correction, as the utilization of extended Hamming codes increases the possibility of recovering individual secret symbols, which all must be recovered independently.

Both methods can be applied to the FD oracle based attack on lattice-based schemes in a non-adaptive attack model. However, the inner-symbol error correction method presented in [Ngo+21] offers no source compression gain and has inferior error correction capabilities. For instance, it is demonstrated in [Ngo+21] that for a platform with an average message bit recovery rate of 0.972, 216 traces, or 9×24, are required to recover the long-term secret key of a masked Saber implementation. We utilize the detailed message bit recovery rates recorded in Table 20 of [Ngo+21] to calculate the corresponding lower bound, which is determined to be 10 traces. This demonstrates a significant gap of 21.6 between the actual performance in a real-world scenario and the calculated lower bound. While there is no guarantee that the non-rigorous lower bound will always be attainable, the small ratio of 1.33, or 12/9, for our SCA-LDPC attack on Kyber, illustrates the superior efficiency of our method in terms of the required number of traces.

The substantial improvement of the new SCA-LDPC framework can be attributed to various factors, such as the utilization of soft information in the real-world attack that we conducted. The dominant reason is that all the secret symbols are interconnected and correlated, and redundant symbols are introduced, allowing for the effective handling of a significant number of symbol-level errors. On the contrary, in the inner-symbol error correction method, all the symbols (e.g., 768 symbols in the Saber case) are independent and needs to be successfully recovered, thus precluding the tolerance of any symbol-level errors. Last, in a real-world attack scenario, several secret positions typically have a higher chance of containing errors, which can be effectively corrected through the inter-symbol approach, but may prove to be a bottleneck for the inner-symbol method where all symbols must be correctly identified independently.

7 Concluding Remarks and Future Work

From coding theory, we have presented a generic framework for key-recovery side-channel attacks on CCA-secure post-quantum encryption/KEM schemes. Our design philosophy is to employ randomly generated LDPC codes with efficient decoding to connect secret coefficients, which introduces additional benefits of source compression and error correction. We presented simulation results and real-world experiments on the main lattice-based KEM Kyber and the code-based KEM HQC. The new attack framework can significantly improve the state-of-the-art in terms of the required number of side-channel measurements. An explanation for the substantial improvements is that LDPC codes are considered to have near-optimal performance from an information-theoretic standpoint.

The sample complexity of the new attack framework can be improved further by (i) employing a more advanced code-construction method with improved decoding performance or by (ii) heavy post-processing such as lattice-reduction or information-set decoding. An intriguing area of study is to utilize sophisticated coding-theoretical methods [RU08], such as density evolution or EXIT charts, to carry out efficient and precise security assessments against proposed attacks.

The new attack framework can be easily applied to several other important KEM candidates in the NIST PQ project such as FrodoKEM [Nae+20] and Saber [DAn+20]. It is interesting to investigate its further applications in NTRU [Che+20] and NTRU prime [Ber+20]. Last, the new attack framework shows the need for countermeasures such as constant-time implementations or higher-order masked implementations. Our next step is to evaluate the security of higher-order masked implementations for Kyber, as Kyber is a future NIST standard. The very recent work [DNG22] demonstrated a high probability of success in recovering message bits from a masked Kyber implementation of up to the fifth order. In conjunction with our simulation results, this suggests higher-order masked implementations may still be highly susceptible to the present attack framework.

References

[Agu+20] C. Aguilar Melchor et al. *HQC*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`. National Institute of Standards and Technology, 2020.

[Alb+20] M. R. Albrecht et al. *Classic McEliece*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`. National Institute of Standards and Technology, 2020.

[Ara+20] N. Aragon et al. *BIKE*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`. National Institute of Standards and Technology, 2020.

[Bac+22] L. Backlund, K. Ngo, J. Gärtner, and E. Dubrova. *Secret Key Recovery Attacks on Masked and Shuffled Implementations of CRYSTALS-Kyber and Saber*. Cryptology ePrint Archive, Paper 2022/1692. `https://eprint.iacr.org/2022/1692`. 2022.

[Ber+20] D. J. Bernstein et al. *NTRU Prime*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`. National Institute of Standards and Technology, 2020.

[Che+20] C. Chen et al. *NTRU*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`. National Institute of Standards and Technology, 2020.

[Dac+20] D. Dachman-Soled, L. Ducas, H. Gong, and M. Rossi. "LWE with side information: attacks and concrete security estimation". In: *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II*. Springer. 2020, pp. 329–358.

[DAn+19] J.-P. D'Anvers, M. Tiepelt, F. Vercauteren, and I. Verbauwhede. "Timing attacks on error correcting codes in post-quantum schemes". In: *Proceedings of ACM Workshop on Theory of Implementation Security Workshop*. 2019, pp. 2–9.

[DAn+20] J.-P. D’Anvers, A. Karmakar, et al. *SABER*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`. National Institute of Standards and Technology, 2020.

[DNG22] E. Dubrova, K. Ngo, and J. Gärtner. *Breaking a Fifth-Order Masked Implementation of CRYSTALS-Kyber by Copy-Paste*. Cryptology ePrint Archive, Paper 2022/1713. 2022.

[FO99] E. Fujisaki and T. Okamoto. “Secure Integration of Asymmetric and Symmetric Encryption Schemes”. In: *Advances in Cryptology – CRYPTO’99*. Ed. by M. J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1999, pp. 537–554.

[For65] G. D. Forney. “Concatenated codes.” In: (1965).

[Gal62] R. Gallager. “Low-density parity-check codes”. In: *IRE Transactions on information theory* 8.1 (1962), pp. 21–28.

[GJJ22] Q. Guo, A. Johansson, and T. Johansson. “A Key-Recovery Side-Channel Attack on Classic McEliece Implementations”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022.4 (2022), pp. 800–827.

[GJN20] Q. Guo, T. Johansson, and A. Nilsson. “A Key-Recovery Timing Attack on Post-quantum Primitives Using the Fujisaki-Okamoto Transformation and Its Application on FrodoKEM”. In: *CRYPTO 2020, Part II*. Ed. by D. Micciancio and T. Ristenpart. Vol. 12171. LNCS. Springer, Heidelberg, Aug. 2020, pp. 359–386.

[GJS16] Q. Guo, T. Johansson, and P. Stankovski. “A Key Recovery Attack on MDPC with CCA Security Using Decoding Errors”. In: *Advances in Cryptology – ASIACRYPT 2016, Part I*. Ed. by J. H. Cheon and T. Takagi. Vol. 10031. Lecture Notes in Computer Science. Hanoi, Vietnam: Springer, Heidelberg, Germany, Dec. 2016, pp. 789–815.

[GLG22] G. Goy, A. Loiseau, and P. Gaborit. “A New Key Recovery Side-Channel Attack on HQC with Chosen Ciphertext”. In: *Post-Quantum Cryptography*. Ed. by J. H. Cheon and T. Johansson. Cham: Springer International Publishing, 2022, pp. 353–371.

[Guo+20] Q. Guo, V. Grosso, F.-X. Standaert, and O. Bronchain. “Modeling Soft Analytical Side-Channel Attacks from a Coding Theory Viewpoint”. In: *IACR TCHES* 2020.4 (2020). `https://tches.iacr.org/index.php/TCHES/article/view/8682`, pp. 209–238.

[Guo+22] Q. Guo, C. Hlauschek, T. Johansson, N. Lahr, A. Nilsson, and R. L. Schröder. "Don't Reject This: Key-Recovery Timing Attacks Due to Rejection-Sampling in HQC and BIKE". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022.3 (2022), pp. 223–263.

[Ham+21] M. Hamburg et al. "Chosen Ciphertext k-Trace Attacks on Masked CCA2 Secure Kyber". In: *IACR TCHES* 2021.4 (2021). `https://tches.iacr.org/index.php/TCHES/article/view/9061`, pp. 88–113.

[Hei+22] D. Heinz, M. J. Kannwischer, G. Land, T. Pöppelmann, P. Schwabe, and D. Sprenkels. *First-Order Masked Kyber on ARM Cortex-M4*. Cryptology ePrint Archive, Paper 2022/058. 2022.

[HGS99] C. Hall, I. Goldberg, and B. Schneier. "Reaction Attacks against several Public-Key Cryptosystems". In: *ICICS 99: 2nd International Conference on Information and Communication Security*. Ed. by V. Varadharajan and Y. Mu. Vol. 1726. Lecture Notes in Computer Science. Sydney, Australia: Springer, Heidelberg, Germany, Nov. 1999, pp. 2–12.

[HHK17] D. Hofheinz, K. Hövelmanns, and E. Kiltz. "A Modular Analysis of the Fujisaki-Okamoto Transformation". In: *TCC 2017: 15th Theory of Cryptography Conference, Part I*. Ed. by Y. Kalai and L. Reyzin. Vol. 10677. Lecture Notes in Computer Science. Baltimore, MD, USA: Springer, Heidelberg, Germany, Nov. 2017, pp. 341–371.

[HSC] S. Huang, R. Q. Sim, and C. Chuengsatiansup. private communication.

[JZ98] T. Johansson and K. S. Zigangirov. "A Simple One-Sweep Algorithm for Optimal APP Symbol Decoding of Linear Block Codes". In: *IEEE Trans. Inf. Theory* 44.7 (1998), pp. 3124–3129.

[Koc96] P. C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In: *Advances in Cryptology – CRYPTO'96*. Ed. by N. Koblitz. Vol. 1109. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1996, pp. 104–113.

[McE78] R. J. McEliece. "A public-key cryptosystem based on algebraic". In: *Coding Thv* 4244 (1978), pp. 114–116.

[Nae+20] M. Naehrig et al. *FrodoKEM*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`. National Institute of Standards and Technology, 2020.

[Ngo+21] K. Ngo, E. Dubrova, Q. Guo, and T. Johansson. "A Side-Channel Attack on a Masked IND-CCA Secure Saber KEM Implementation". In: *IACR TCHES* 2021.4 (2021). `https://tches.iacr.org/index.php/TCHES/article/view/9079`, pp. 676–707.

[NIS18] NIST. *NIST Post-Quantum Cryptography Standardization*. `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization` Accessed: 2018-09-24. 2018.

[Pea82] J. Pearl. "Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach". In: *AAAI*. 1982.

[PPM17] R. Primas, P. Pessl, and S. Mangard. "Single-trace side-channel attacks on masked lattice-based encryption". In: *Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*. Springer. 2017, pp. 513–533.

[Qin+21] Y. Qin, C. Cheng, X. Zhang, Y. Pan, L. Hu, and J. Ding. "A Systematic Approach and Analysis of Key Mismatch Attacks on Lattice-Based NIST Candidate KEMs". In: *ASIACRYPT 2021, Part IV*. Ed. by M. Tibouchi and H. Wang. Vol. 13093. LNCS. Springer, Heidelberg, Dec. 2021, pp. 92–121.

[Raj+22] G. Rajendran, P. Ravi, J.-P. D'Anvers, S. Bhasin, and A. Chattopadhyay. *Pushing the Limits of Generic Side-Channel Attacks on LWE-based KEMs - Parallel PC Oracle Attacks on Kyber KEM and Beyond*. Cryptology ePrint Archive, Paper 2022/931. 2022.

[Rav+20] P. Ravi, S. S. Roy, A. Chattopadhyay, and S. Bhasin. "Generic Side-channel attacks on CCA-secure lattice-based PKE and KEMs". In: *IACR TCHES* 2020.3 (2020). `https://tches.iacr.org/index.php/TCHES/article/view/8592`, pp. 307–335.

[Rav+22] P. Ravi, M. F. Ezerman, S. Bhasin, A. Chattopadhyay, and S. S. Roy. "Will You Cross the Threshold for Me? Generic Side-Channel Assisted Chosen-Ciphertext Attacks on NTRU-based KEMs". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022.1 (2022), pp. 722–761.

[Reg05] O. Regev. "On lattices, learning with errors, random linear codes, and cryptography". In: *37th Annual ACM Symposium on Theory of Computing*. Ed. by H. N. Gabow and R. Fagin. Baltimore, MA, USA: ACM Press, May 2005, pp. 84–93.

[RR] P. Ravi and S. S. Roy. *Side-Channel Analysis of Lattice-based PQC Candidates.* `https://csrc.nist.gov/CSRC/media/Projects/post-quantum-cryptography/documents/round-3/seminars/mar-2021-ravi-sujoy-presentation.pdf`. Accessed: 2022-09-29.

[RSU01] T. Richardson, M. Shokrollahi, and R. Urbanke. "Design of capacity-approaching irregular low-density parity-check codes". In: *IEEE Transactions on Information Theory* 47.2 (2001), pp. 619–637.

[RU08] T. Richardson and R. Urbanke. *Modern Coding Theory.* USA: Cambridge University Press, 2008.

[Sch+20] P. Schwabe et al. *CRYSTALS-KYBER.* Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`. National Institute of Standards and Technology, 2020.

[Sch+22] T. Schamberger, L. Holzbaur, J. Renner, A. Wachter-Zeh, and G. Sigl. *A Power Side-Channel Attack on the Reed-Muller Reed-Solomon Version of the HQC Cryptosystem.* Cryptology ePrint Archive, Paper 2022/724. 2022.

[She+22] M. Shen, C. Cheng, X. Zhang, Q. Guo, and T. Jiang. *Find the Bad Apples: An efficient method for perfect key recovery under imperfect SCA oracles – A case study of Kyber.* Cryptology ePrint Archive, Paper 2022/563. 2022.

[Sho94] P. W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *35th Annual Symposium on Foundations of Computer Science.* Santa Fe, NM, USA: IEEE Computer Society Press, Nov. 1994, pp. 124–134.

[Tan+22] Y. Tanaka, R. Ueno, K. Xagawa, A. Ito, J. Takahashi, and N. Homma. *Multiple-Valued Plaintext-Checking Side-Channel Attacks on Post-Quantum KEMs.* Cryptology ePrint Archive, Paper 2022/940. 2022.

[Uen+22] R. Ueno, K. Xagawa, Y. Tanaka, A. Ito, J. Takahashi, and N. Homma. "Curse of Re-encryption: A Generic Power/EM Analysis on Post-Quantum KEMs". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022.1 (2022), pp. 296–322.

[VGS14] N. Veyrat-Charvillon, B. Gérard, and F.-X. Standaert. "Soft Analytical Side-Channel Attacks". In: *ASIACRYPT 2014, Part I.* Ed. by P. Sarkar and T. Iwata. Vol. 8873. LNCS. Springer, Heidelberg, Dec. 2014, pp. 282–296.

[Xu+22] Z. Xu, O. Pemberton, S. S. Roy, D. Oswald, W. Yao, and Z. Zheng. “Magnifying Side-Channel Leakage of Lattice-Based Cryptosystems With Chosen Ciphertexts: The Case Study of Kyber”. In: *IEEE Transactions on Computers* 71.9 (2022), pp. 2163–2176.

Appendix A: Supporting Figures and Tables

Algorithm 3 Kyber.CPAPKE.KeyGen()

```
Input:
Output: $sk, pk$
  $\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{d\times d}$
  $\mathbf{s} \xleftarrow{\$} \mathsf{B}_{\mu_1}^d$, where $\mathbf{s} \in \mathcal{R}_q^d$
  $\mathbf{e} \xleftarrow{\$} \mathsf{B}_{\mu_1}^d$, where $\mathbf{e} \in \mathcal{R}_q^d$
  $sk \overset{\text{def}}{=} \mathbf{s}$
  $pk \overset{\text{def}}{=} \mathbf{A}\mathbf{s} + \mathbf{e}$
```

Algorithm 4 Kyber.CPAPKE.Dec()

```
Input: $sk, c = (\mathbf{c}_1, c_2)$
Output: m
  $\mathbf{u} = \mathtt{Decomp}_q(\mathbf{c}_1, d_u)$
  $v = \mathtt{Decomp}_q(c_2, d_v)$
  $m = \mathtt{Comp}_q(v - \mathbf{s}^{\mathrm{T}}\mathbf{u}, 1)$
```

Algorithm 5 Kyber.CPAPKE.Enc()

```
Input: $pk, m, r$
Output: $c = (\mathbf{c}_1, c_2)$
  Generate $\mathbf{A} \in \mathcal{R}_q^{d\times d}$ from $pk$
  $\mathbf{p} = pk$
  $\mathbf{r} \xleftarrow{\$} \mathsf{B}_{\mu_1}^d$, where $\mathbf{r} \in \mathcal{R}_q^d$
  $\mathbf{e}_1 \xleftarrow{\$} \mathsf{B}_{\mu_2}^d$, where $\mathbf{e}_1 \in \mathcal{R}_q^d$
  $e_2 \xleftarrow{\$} \mathsf{B}_{\mu_2}$, where $e_2 \in \mathcal{R}_q$
  $\mathbf{u} = \mathbf{A}^{\mathrm{T}}\mathbf{r} + \mathbf{e}_1$
  $v = \mathbf{p}^{\mathrm{T}}\mathbf{r} + e_2 + \mathtt{Decomp}_q(m, 1)$
  $\mathbf{c}_1 \overset{\text{def}}{=} \mathtt{Comp}_q(\mathbf{u}, d_u)$
  $c_2 \overset{\text{def}}{=} \mathtt{Comp}_q(v, d_v)$
  $c \overset{\text{def}}{=} (\mathbf{c}_1, c_2)$
```

Figure 3: Kyber.CPAPKE (simplified version)

Algorithm 6 Kyber.CCAKEM.KeyGen()

```
Input:
Output: $sk, pk$
  Generate a pseudo-random coin $z$
  $(pk, sk') \overset{\text{def}}{=}$ Kyber.CPAPKE.KeyGen()
  $sk \overset{\text{def}}{=} (sk', pk, \mathcal{H}(pk), z)$
```

Algorithm 7 Kyber.CCAKEM.Encaps()

```
Input: $pk$
Output: $c, K$
  $m \xleftarrow{\$}{}^{256}$
  $m = \mathcal{H}(m)$
  $(\bar{K}, r) = \mathcal{G}(m, \mathcal{H}(pk))$
  $c =$ Kyber.CPAPKE.Enc$(pk, m, r)$
  $K =$ KDF$(\bar{K}, \mathcal{H}(c))$
```

Algorithm 8 Kyber.CCAKEM.Decaps()

```
Input: $sk, c$
Output: $K$
  $m' =$ Kyber.CPAPKE.Dec$(sk, c)$
  $(\bar{K}', r') = \mathcal{G}(m', \mathcal{H}(pk))$
  $c' =$ Kyber.CPAPKE.Enc$(pk, m', r')$

  if $c = c'$ then
    $K =$ KDF$(\bar{K}', \mathcal{H}(c))$
  else
    $K =$ KDF$(z, \mathcal{H}(c))$
  end if
```

Figure 4: Kyber.CCAKEM

- Setup(1^λ): generates the global parameters param $= (n, k, \delta, \omega, \omega_{\mathbf{r}}, \omega_{\mathbf{e}})$.

- KeyGen(param): sample $\mathbf{h} \xleftarrow{\$} \mathcal{R}_2$, the generator matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$ of $\mathcal{C}$, sk $= (\mathbf{x}, \mathbf{y}) \xleftarrow{\$} \mathcal{R}_2^2$ such that $\omega(\mathbf{x}) = \omega(\mathbf{y}) = \omega$, sets pk $= (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$, and returns (pk, sk).

- Encrypt(pk, $\mathbf{m}$): generates $\mathbf{e} \xleftarrow{\$} \mathcal{R}_2$, $\mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2) \xleftarrow{\$} \mathcal{R}_2^2$ such that $\omega(\mathbf{e}) = \omega_{\mathbf{e}}$ and $\omega(\mathbf{r}_1) = \omega(\mathbf{r}_2) = \omega_{\mathbf{r}}$, sets $\mathbf{u} = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2$ and $\mathbf{v} = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}$, returns $\mathbf{c} = (\mathbf{u}, \mathbf{v})$.

- Decrypt(sk, $\mathbf{c}$): returns $\mathcal{C}$.Decode$(\mathbf{v} - \mathbf{u} \cdot \mathbf{y})$.

Figure 5: Description of the proposal HQC.PKE [Agu+20].

- $\mathsf{Setup}(1^\lambda)$: generates the global parameters $\mathsf{param} = (n, k, \delta, \omega, \omega_\mathbf{r}, \omega_\mathbf{e})$.

- $\mathsf{KeyGen}(\mathsf{param})$: exactly as in $\mathsf{HQC.PKE}$.

- $\mathsf{Encapsulate}(\mathsf{pk})$: generate $\mathbf{m} \xleftarrow{\$} \mathbb{F}_2^k$, which will serve as the seed to derive the shared key. Derive the randomness $\theta \xleftarrow{\$} \mathcal{G}(\mathbf{m})$. Generate the ciphertext $\mathbf{c} \leftarrow (\mathbf{u}, \mathbf{v}) = \mathcal{E}.\mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}, \theta)$, and derive the symmetric key $K \leftarrow \mathcal{K}(\mathbf{m}, \mathbf{c})$. Let $\mathbf{d} \leftarrow \mathcal{H}(\mathbf{m})$, and send $(\mathbf{c}, \mathbf{d})$.

- $\mathsf{Decapsulate}(\mathsf{sk},\mathbf{c},\ \mathbf{d})$: decrypt $\mathbf{m}' \leftarrow \mathcal{E}.\mathsf{Decrypt}(\mathsf{sk}, \mathbf{c})$, compute $\theta' \leftarrow \mathcal{G}(\mathbf{m}')$, and (re-)encrypt $\mathbf{m}'$ to get $\mathbf{c}' \leftarrow \mathcal{E}.\mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}', \theta')$. If $\mathbf{c} \neq \mathbf{c}'$ or $\mathbf{d} \neq \mathcal{H}(\mathbf{m}')$ then abort. Otherwise, derive the shared key $K \leftarrow \mathcal{K}(\mathbf{m}, \mathbf{c})$.

Figure 6: Description of the proposal $\mathsf{HQC.KEM}$ [Agu+20].

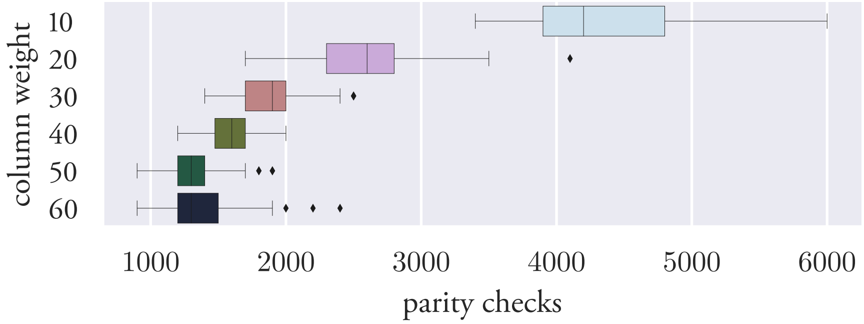


Figure 7: Experimental results of varying the column weight of the generated LDPC code. Weight of 50 appears to require the fewest amount of parity checks for successful key recovery.

Table 8: The best inner codes for secret coefficients for given accuracy level ρ.

ρ	m_0	(k'_u, k'_v)	Secret coefficient				
			-2	-1	0	1	2
$0.995, 0.95$	2	$(630, 0)$	0	1	0	1	0
		$(706, 6)$	0	0	1	1	0
$1, 0.95$	3	$(630, 14)$	0	1	0	1	1
		$(706, 6)$	0	0	1	1	0
		$(706, 10)$	0	1	1	0	0
0.9	4	$(630, 14)$	0	1	0	1	1
		$(706, 6)$	0	0	1	1	0
		$(706, 10)$	0	1	1	0	0
		$(630, 10)$	0	0	1	0	1

Table 9: "Reasonable" inner codes for check variables. We assume that each check variable is the sum of 6 secret coefficients.

m_2	(k'_u, k'_v)	Coefficients for check variable, $[-12, \ldots, 12]$
2	$(180, 0)$	0001100011100011100011000
	$(423, 14)$	0101001010110101001010110
3	$(401, 1)$	0101001010010110101101001
	$(630, 11)$	0100101001011010010100101
	$(483, 7)$	0101011010101010101010100
4	$(636, 5)$	0010110100101101001011010
	$(486, 1)$	0101010101010101010110101
	$(139, 2)$	0111000011110001111000011
	$(630, 9)$	0110101101001011010110100

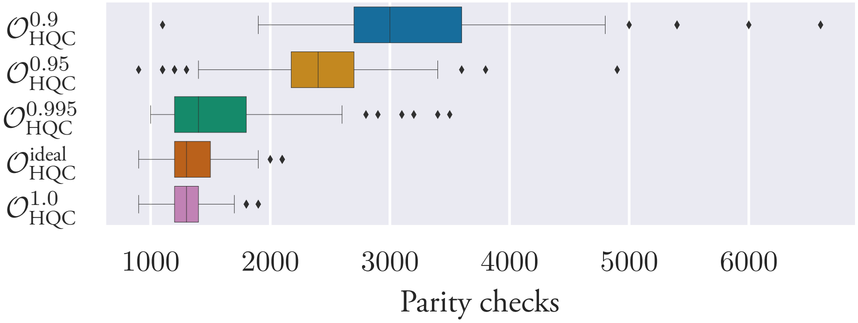


Figure 8: Boxplot of Table 11

Table 10: The architecture of the neural network used for the real-world attack on Kyber.

Layer type	(Input, output) shape	# Parameters
Batch Normalization 1	(64, 64)	256
Dense 1	(64, 64)	4160
Batch Normalization 2	(64, 64)	256
ReLU	(64, 64)	0
Dense 2	(64, 32)	2080
Batch Normalization 3	(32, 32)	128
ReLU	(32, 32)	0
Dense 3	(32, 16)	528
Batch Normalization 3	(16, 16)	64
ReLU	(16, 16)	0
Dense 4	(16, 1)	17
Sigmoid	(1, 1)	0

Table 11: HQC number of required parity checks for successful decoding

ρ	weight	count	mean	std	min	25%	50%	75%	max
$\mathcal{O}_{\mathrm{HQC}}^{0.9}$	50	103	3222.33	873.57	1100	2700	3000	3600	6600
$\mathcal{O}_{\mathrm{HQC}}^{0.95}$	50	100	2396	629.40	900	2175	2400	2700	4900
$\mathcal{O}_{\mathrm{HQC}}^{0.995}$	50	100	1623	546.40	1000	1200	1400	1800	3500
$\mathcal{O}_{\mathrm{HQC}}^{\mathrm{ideal}}$	50	110	1365.45	261.41	900	1200	1300	1500	2100
$\mathcal{O}_{\mathrm{HQC}}^{1.0}$	50	100	1324	208.47	900	1200	1300	1400	1900

Popular Science Summary in Swedish

Populärvetenskaplig sammanfattning

Kvantdatorer bedöms kunna ha kapacitet att knäcka kryptering, även om risken ligger ett antal utvecklingsår framåt i tiden. Det är ju bra för oss som bryr oss om säkerhet, men det tar lång tid att bl.a. designa, utvärdera, standardisera, implementera och distribuera nya kryptalgoritmer. Därför är det av största vikt att det arbetet startar i god tid.

En säker krypteringsalgoritm utgår från en solid teoretisk säkerhetsgrund, men vid implementation i mjukvara och/eller hårdvara riskerar man introduktion av nya sårbarheter. Algoritmens inre tillstånd kan nämligen påverkas eller läckas genom olika attacktekniker.

Sidokanalsattacker (eng. side-channel attacks, SCA) är en gren inom kryptanalys och är ett viktigt ben som denna avhandling vilar på. Vid SCA kan man mäta t.ex. tidsvariationer för att nyttja beroenden mellan indata och tillstånd, eller utläsa olika exekveringsvägar inom algoritmen. Detta är ett väldigt kraftfullt verktyg för en angripare, då en upptäckt sårbarhet har god chans att kunna nyttjas över nätverk, och utan fysisk tillgång till det angripna målet. Vissa tidsvariationer har sitt ursprung i algoritmernas uppbyggnad, d.v.s. beroende på hur den hemliga nyckeln ser ut kommer beräkningar utföras på olika sätt. Andra tidsvariationer uppstår vid översättningen från teori till maskinkod.

SCA kan även orsakas av strömförbrukningsvariationer eller variationer av elektromagnetisk strålning. I dessa fall krävs fysisk tillgång för att kunna utföra mätningar. Många användarfall placerar hårdvara som t.ex. smarta kort eller Internet-Of-Things (IOT) i exponerade miljöer och dessa bör därför vara skyddade, även mot sådana fysiska attacker.

Denna avhandling handlar i huvudsak om hur insamling av dekrypteringsfel kan ge en angripare tillräckligt med information för att knäcka antingen den hemliga nyckeln eller ett hemligt meddelande. För många krypteringsalgoritmer finns det nämligen en gemensam egenskap. De har alla en osannolik, men ändå större än noll, risk för att dekryptering skall misslyckas för korrekt krypterade meddelanden. Kända krypteringsalgoritmer har valt parametrar så att risken är minimal, men som forskningen i denna avhandling visar, kan denna egenskap fortfarande orsaka problem, i vissa fall.

Denna avhandling fokuserar på skärningspunkten av implementationsaspekter, kryptanalys, kodningsteori och lite gitter-baserad kryptografi, enligt nedanstående beskrivningar.

Implementationsaspekter syftar till forskning på upptäckta brister och sidokanalsläckage i mjukvaruimplementationerna av kryptalgoritmer. Forskningen inkluderar även en alternativ implementation av en specifik del av en krypteringsalgoritm.

Kryptanalys innebär försök att förbättra förståelsen för nästa generation av kryptalgoritmer. I några fall upptäckte vi nya sårbarheter i källkoden, i andra fall publicerade vi nya strategier för att förbättra redan existerande attacker. Här återfinns även nya teoretiska resultat kring den praktiska säkerhetsnivån för några av nästa generations krypteringsalgoritmer, i relation till dekrypteringsfel, som nämnts ovan.

Kodningsteori relaterar till teori kring felrättande koder, antingen som ett analytiskt verktyg, eller för att de berör kodbaserade krypteringsalgoritmer. Felrättande koder används framförallt för att skicka och ta emot signaler över olika medium, men kan även användas inom kryptografi då felkorrigering av slumpmässiga och ostrukturerade koder är ett beräkningsmässigt svårt problem. Detta kan tillämpas som en god teoretisk grund för säkerheten för vissa av nästa generationens krypteringsalgoritmer.

Gitter-baserad kryptering är en viktig del av denna avhandling då några algoritmer av denna typ utsätts för varierande grad av granskning. Gitter (eng. Lattice) kan liknas vid N-dimensionella koordinatsystem med endast heltalskoordinater, där N i kryptosammanhang är ett stort tal. Kring dessa matematiska konstruktioner kretsar ett antal problem, vars lösningar är mycket tunga att beräkna. Detta ligger till grunden för säkerheten för många av nästa generations krypteringsalgoritmer.

"Attacker blir bara bättre, inte sämre" är en välkänd truism inom fältet kryptanalys. Detta är ett viktigt forskningsområde eftersom den praktiska tillämparheten ofta används som en måttstock över hur mycket kraft som skall spenderas på att förebygga och mitigera attacker. Nya, tidigare okända, attacker fyller samma syfte, fast med något större genomslag i det kryptanalytiska forskningsområdet. Det beror naturligtvis på att implementationsproblem bara kan fixas om de är kända. I denna avhandling gav vi också ett förslag till en alternativ subkomponent till en redan känd kryptalgoritm. Detta ifrågasätter "status quo" och driver innovation, även om det specifika alternativet i fråga inte har kommit till någon vidare användning. Slutligen, finner vi även forskningsbidrag som ligger mer åt det teoretiska hållet och medan det är användbart i sig själv så bidrar det även rent generellt till den större massan av kryptanalytisk kunskap. Detta höjer förtroendet för den viktiga säkerhetsutvärderingen av nästa generation av kvantdatorsäkra krypteringsalgoritmer.