Attacking and defending post-quantum cryptography candidates

Thales Areco Bandiera Paiva

Tese apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Doutor em Ciências

Programa: Ciência da Computação
Orientador: Prof. Dr. Routo Terada

Durante o desenvolvimento deste trabalho, o autor recebeu auxílio financeiro da CAPES

São Paulo, janeiro de 2023

Attacking and defending
post-quantum cryptography candidates

Esta versão da dissertação contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 17/11/2022. Uma cópia da versão original está disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Prof. Dr. Routo Terada (orientador) – IME–USP
- Prof. Dr. Marcos Simplício Jr. – EP–USP
- Profa. Dra. Denise Goya – CMCC–UFABC
- Prof. Dr. Julio López – IC–Unicamp
- Dr. Rafael Misoczki – Google

Agradecimentos

Ao Professor Routo pelos anos trabalhando juntos, sempre com muita leveza. Aos membros das comissões julgadoras, tanto a de defesa quanto a de qualificação, pela atenção e disponibilidade em nos ajudar a melhorar este trabalho. À Ali, à família e aos amigos pelo apoio. Aos colegas de criptografia pelas discussões.

Este trabalho é dedicado à Professora Maria Lúcia Diniz, minha grande amiga.

Resumo

Paiva, T. A. B. **Atacando e defendendo esquemas criptográficos pós-quânticos**. Tese - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2022.
Em criptografia pós-quântica, estamos interessados em esquemas criptográficos baseados em problemas cuja solução, acredita-se, não pode ser encontrada eficientemente nem com o uso de computadores quânticos. Esta dissertação, escrita no formato de coletânea de artigos, apresenta contribuições originais sobre a segurança e implementação de três candidatos a esquemas pós-quânticos: HQC, PKP e BIKE. Ambos HQC e BIKE são esquemas de encapsulamento de chaves (KEM) baseados em códigos corretores de erros que foram selecionados pelo NIST como candidatos alternativos em seu processo de padronização de esquemas pós-quânticos. O problema do núcleo permutado (PKP) é um problema NP-difícil que pode ser usado para instanciar esquemas pós-quânticos de assinaturas digitais. A primeira contribuição é um ataque ao HQC usando informações sobre o tempo de execução do algoritmo de encriptação. Este ataque permite a um atacante recuperar a chave privada de uma vítima após medir o tempo de execução de 400 milhões de operações de decriptação, considerando parâmetros para 128 bits de segurança. A segunda contribuição consiste no primeiro ataque a uma generalização do PKP para corpos pequenos. Para parâmetros que prometem 80 bits de segurança, o ataque recupera uma fração de 2^{-40} das chaves com apenas 2^{48} operações, e aproximadamente 7.2% das chaves com 2^{62} operações. A terceira e última contribuição consiste num novo algoritmo de decriptação para o BIKE. O algoritmo foi implementado em tempo constante e observou-se speedups de 1,18, 1,29 e 1,47 em relação ao estado da arte, considerando os níveis de segurança 128, 192, e 256, respectivamente.
Palavras-chave: Criptografia pós-quântica, BIKE, HQC, PKP, ataque por tempo de execução, implementação em tempo constante, criptanálise

Abstract

Paiva, T. A. B. **Attacking and defending post-quantum cryptography candidates**. Dissertation – Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2022.

In Post-Quantum Cryptography, we are interested in schemes based on problems which are believed to be hard even for quantum computers. This dissertation, which is written as a collection of papers, presents original contributions to the security and implementation of three post-quantum cryptography candidates: HQC, PKP and BIKE. Both HQC and BIKE are code-based key encapsulation mechanisms that were selected as alternate candidates in NIST's post-quantum standardization process. The Permuted Kernel Problem (PKP) is an NP-hard combinatorial problem that can be used to instantiate post-quantum digital signature schemes. The first contribution is a timing attack against HQC that allows an attacker to recover the secret key after recording the decryption time of around 400 million ciphertexts, for 128 bits of security. The second contribution consists of the first attack targeting a generalization of PKP for small fields. For 80-bit security parameters, the attack is able to recover a fraction 2^{-40} of the keys using only 2^{48} operations, and about 7.2% of the keys using 2^{62} operations. The third and last contribution consists of a new decryption algorithm for BIKE. Our constant-time implementation of this algorithm achieves speedups of 1.18, 1.29 and 1.47, with respect to state-of-the-art decryption algorithms, for security levels 128, 192 and 256, respectively.
Keywords: Post-quantum cryptography, BIKE, HQC, PKP, timing attack, constant-time implementation, cryptanalysis

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In 1994, Shor [Sho94] published two algorithms for quantum computers that efficiently solve two critical problems in modern cryptography: the discrete logarithm and the integer factorization problems. These problems are the building blocks of the most common public key schemes used today such as Elliptic Curves [Mil86] and RSA [RSA78], both for encryption and digital signatures. With the development of larger and more powerful quantum computers, important protocols for secure communication such as TLS [Res18], SSH [Ylo96], and Signal[1] are at risk of being broken.

In the last few years, we saw significant efforts by companies like Google, IBM, and D–Wave in building and programming quantum computers. Fortunately, there are still some major engineering problems that must be solved before we can see the complete cryptanalysis of public-key schemes. It is difficult to estimate if and when quantum computers will be able to attack real-world parameters. An estimate that is often used is the one by Mosca[2][Mos18], who estimates as 1/7 the probability of quantum computers breaking RSA with 2048 bits by 2026, and 1/2 by the end of 2031.

Although one can argue against Mosca's hypothesis and estimates, it is getting increasingly difficult to reasonably argue against the idea of basing real-world cryptography in problems that are not known to be broken by quantum computers. Given the scale of the mass surveillance carried out by agencies such as NSA and GCHQ revealed by Snowden, one major threat against today's privacy are attacks where an agency stores encrypted communications now to decrypt in the future, when a sufficiently large quantum computer is available [JMM+22].

Post-quantum cryptography (PQC) [BBD09] defines models, schemes, and protocols that are based on problems that are not known to be broken by quantum computers. It is important to note that the security of public-key and symmetric-key cryptography against quantum computers are not equally affected. For symmetric-key cryptography, the main threat is Grover's [Gro96] algorithm. When applied to symmetric-key cryptography, this algorithm provides a quadratic speedup in a brute-force search for the secret key.

[1] https://github.com/signalapp/libsignal

[2] Michele Mosca is the deputy director of the Institute for Quantum Computing at the University of Waterloo.

Therefore, the approach taken for post-quantum symmetric-key cryptography is rather straightforward: double the secret key size.

For public-key cryptography, we need to construct schemes based on problems that, unlike factoring and computing discrete logarithms, are believed to be hard even for post-quantum computers. This includes some problems from Coding Theory, Lattices, Multivariate Quadratic Equations, and even Hash Functions. While there are post-quantum (public-key) encryption schemes as old as RSA, such as the McEliece scheme [McE78], the key sizes and efficiency of post-quantum schemes are typically worse than the ones for Elliptic Curves, when comparing parameters achieving the same level of security against classical[3] adversaries.

Since 2016, the National Institute of Standards and Technology (NIST) is running a standardization process for post-quantum cryptography [CCJ+16]. This process aims to standardize post-quantum Key Encapsulation Mechanism (KEM) and Digital Signature Schemes (DSS). These are important building blocks for secure communication. DSS are used to ensure authenticity and integrity, and KEMs[4] are used to exchange a shared key that will be used for communication using symmetric-key schemes such as AES. A similar standardization initiative was conducted by the Chinese Association for Cryptographic Research (CACR), but there is little information about this process outside China.[5] Therefore, the overview presented here is highly biased towards NIST's PQC process.

NIST's process was a turning point for post-quantum cryptography – it doubled as a powerful validation of the practical importance of post-quantum cryptography schemes and as an incentive to design and implement new efficient schemes. As a consequence, the concentrated effort helped a number of both theoretical and practical results to flourish. Not only new cryptographic schemes were constructed, but also new cryptographic models considering quantum adversaries [BHH+19, HHK17a] and the cryptanalysis of important schemes [APRS20, Beu22, CD22] that were believed to be secure.

The first round of NIST's PQC initiative had a large number of submissions. After each round, NIST selects a number of candidates that will move to the next round based on a number of factors, for example: new attacks discovered between rounds, public key and signatures sizes, efficiency, and difficulty of constant-time efficient implementation. NIST also makes it clear that they aim for diversity of standardized schemes with respect to the underlying problems.

Recently, NIST released the report on the third round of its PQC process [AAC+22]. The decision for KEMs and signatures are shown in Table 1.1 and Table 1.2, respectively. We can see that one KEM and 3 signature schemes were already selected for standardization. There are, however, 4 KEMs that are still being considered for standardization but for which NIST will require at least one more round of evaluation. That is, in the future, NIST may decide to also standardize some of the schemes that advanced to the

[3] That is, adversaries running classical, non-quantum, algorithms.

[4] Intuitively, we can think of KEMs as a form of public-key encryption scheme that encrypts a fresh randomly generated key every time.

[5] Most of the information on CACR's initiative is in Chinese.

next round. It is important to note that, shortly after NIST's report was published, a devastating attack against isogeny-based KEMs was published [CD22], which will likely impact its consideration during the next round.

Key encapsulation mechanism	Underlying problem	NIST's third round decision
CRYSTALS–Kyber [ABD+19]	Lattices	Selected for standardization
Classic McEliece [BCL+19]	Codes	Selected for next round
BIKE [ABB+21]	Codes	
HQC [MAB+18]	Codes	
SIKE [ACC+17]	Isogenies	

Table 1.1: Proposals for Key Encapsulation Mechanisms selected by NIST in the third round of its post-quantum standardization process.

Digital Signature Scheme	Underlying problem	NIST's third round decision
CRYSTALS–Dilithium [DKL+18]	Lattices	Selected for standardization
Falcon [FHK+18]	Lattices	
SPHINCS+ [ABB+19]	Hashes	

Table 1.2: Proposals for Digital Signatures selected by NIST in the third round of its post-quantum standardization process.

1.1 The focus of this dissertation

The main focus of this dissertation is on evaluating the security of candidates for post-quantum cryptography. More specifically, we are interested in side-channel attacks, classical cryptanalysis and secure implementation of the candidate schemes. The three schemes for which we identified research opportunities and managed to improve the state of the art are HQC [MAB+18], BIKE [ABB+21], and signatures based on the Permuted Kernel Problem when defined over binary fields [LP11].

BIKE [ABB+21] is a code-based KEM that was recently selected to move to the fourth round of NIST's PQC standardization process. This KEM is based on the Niederreiter scheme instantiated with QC-MDPC codes, and it uses the BGF decoder for key decapsulation.

HQC [MAB+18], which stands for Hamming Quasi-Cyclic, is a KEM based on the hardness of the quasi-cyclic syndrome decoding problem, a conjectured hard problem from Coding Theory. HQC is a candidate for the NIST standardization process that offers reasonably good parameters, with smaller key sizes than the classic McEliece scheme [McE78, BCL+19, ACP+18], but without relying on codes with a secret sparse structure, such as the QC-MDPC [MTSB13] codes used by [ABB+21], or QC-LDPC [Bal14] used by [APRS20].

The Permuted Kernel Problem (PKP) is an NP-hard combinatorial problem [GJ79] that can be used to instantiate post-quantum signature schemes. One instantiation, known

as PKP-DSS [BFK+19] was a finalist of the CACR post-quantum standardization process. The signature schemes are obtained by applying the Fiat-Shamir transform over Shamir's PKP-based identification scheme [Sha89].

1.2 Contributions

This dissertation is based on three papers containing independent contributions related to the schemes listed in the previous section. Each paper and their corresponding contributions are listed below, in chronological order.

1.2.1 A timing attack on HQC

In 2019, we noticed one problem of the HQC's [MAB+18] reference implementation submitted to NIST in the first round: one important component of the decryption operation, namely, the BCH decoding, was not implemented in constant-time. We showed how this could be exploited to mount the first key-recovery timing attack against HQC's implementation. The attack is practical, requiring the attacker to record the decryption time of around 400 million ciphertexts for a set of HQC parameters corresponding to 128 bits of security.

This work was published as the following conference paper:

- Thales Bandiera Paiva and Routo Terada. A timing attack on the HQC encryption scheme. In Kenneth G. Paterson and Douglas Stebila, editors, *Selected Areas in Cryptography – SAC 2019*, pages 551–573, Cham, 2020. Springer International Publishing.

1.2.2 An attack against the binary variant of PKP

In this paper, we propose the first attack that targets the binary PKP [LP11, LP12]. The attack is analyzed in detail, and its practical performance is compared with our theoretical models. For the proposed parameters originally targeting 79 and 98 bits of security, our attack can recover about 100% of all keys using less than 2^{63} and 2^{77} operations, respectively.

This work was published as the following conference paper:

- Thales Bandiera Paiva and Routo Terada. Cryptanalysis of the binary permuted kernel problem. In *International Conference on Applied Cryptography and Network Security – ACNS 2021*, pages 396–423. Springer, 2021.

1.2.3 A novel decryption algorithm for BIKE

We discovered important limitations of BIKE's [ABB+21] decryption algorithm, known as BGF [DGK20c]. This algorithm is analyzed in detail, and then we propose a new decoding algorithm for QC-MDPC codes called PickyFix to address the limitations that were

observed. Our decoder uses two auxiliary iterations that are significantly different from previous approaches and we show how they can be implemented efficiently. We analyze our decoder with respect to both its error correction capacity and its performance in practice. When compared to BGF, our constant-time implementation of PickyFix achieves speedups of 1.18, 1.29, and 1.47 for the security levels 128, 192 and 256, respectively.

This work was published as the following journal paper:

- Thales Bandiera Paiva and Routo Terada. Faster constant-time decoder for MDPC codes and applications to BIKE KEM. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022.

1.2.4 Contributions outside the scope of this dissertation

During my PhD, I was also involved in the following works on computer and network security, but they are outside the scope of this dissertation.

- Thales Bandiera Paiva, Javier Navaridas, and Routo Terada. Robust covert channels based on DRAM power consumption. In *International Conference on Information Security*, pages 319–338. Springer, 2019.
- Thales Bandiera Paiva, Yaissa Siqueira, Daniel Macêdo Batista, Roberto Hirata, and Routo Terada. BGP anomalies classification using features based on AS relationship graphs. In *2021 IEEE Latin-American Conference on Communications (LATINCOM)*, pages 1–6. IEEE, 2021.

Until 2020, we expected that the first of these papers would also be part of this dissertation, as it was a first step towards using the Intel RAPL registers for power-based side-channel attacks against cryptographic schemes. However, in 2021, an attack exploiting these registers for this purpose was published at IEEE S&P [LKO+21]. Then, since our paper is not directly related to cryptography, we decided not to include it in the final version of this dissertation.

1.3 Organization

This dissertation is written as a collection of the three papers mentioned in the previous section. We begin by providing some general background on essential cryptography concepts in Chapter 2. The timing attack on HQC is presented in Chapter 3. Next, in Chapter 4, the cryptanalysis on the binary PKP is detailed. Chapter 5 presents the last paper on an efficient decryption algorithm for BIKE. The dissertation ends with a brief discussion in Chapter 6.

Chapter 2

Background in cryptography and coding theory

Since the next chapters are based on research papers published in cryptography venues, it is often assumed that the reader is familiar with most of the jargon used in cryptography. This chapter provides definitions and explanations of the cryptography concepts that are used, but which are not defined, in the next chapters.

The idea is to introduce the concepts in an intuitive way with the hope that a newcomer to the field can get the most out of this dissertation without getting lost into a large number of formal definitions. For a more formal and in-depth treatment of cryptography, the reader may consider Katz and Lindell's book [KL21]. For coding theory, books such as the ones by van Tilborg[vT93] and Ryan and Lin [RL09] are fine references.

2.1 General definitions

This section presents some essential concepts in modern cryptography such as how to measure the security of cryptographic schemes and the objectives that these schemes help us achieve. Additionally, primitives such as symmetric-key cryptography and hash functions, are briefly reviewed. Although, at first sight, these primitives do not appear to be directly related to public-key cryptography, which is the focus of this dissertation, they are essential for the two following reasons.

1. Hash functions play a crucial role when building secure and efficient public-key schemes.

2. Symmetric-key cryptography is often combined with public-key schemes, which is the main motivation for the post-quantum key encapsulation mechanisms studied in this work.

2.1.1 Basic cryptography objectives

There are many security objectives required by real-world applications [GT11]. Some common requirements may be confidentiality or authenticity, but more complex applications may require some degree of anonymity or more complex privacy requirements. Below we review the most common security objectives that are achievable with simple cryptographic primitives, such as symmetric-key encryption and message authentication codes (MAC) or public-key encryption and signature schemes.

1. **Confidentiality** means that only the intended receiver of a message can read it.
2. **Authenticity** of a message implies that it was genuinely generated by a given sender or group of senders.
3. **Integrity** is the property that guarantees that the message was not manipulated while being transmitted from the sender to the intended receiver.
4. **Nonrepudiation** guarantees that the original sender of a message cannot deny that they were responsible for its creation.

Confidentiality is typically associated with encryption, while authenticity and integrity can be achieved with message authentication codes and digital signatures. Nonrepudiation requires public-key primitives to be achieved, and therefore digital signatures are the typical way in which this objective is reached.

It is important to note that the security requirements for securing real-world systems highly depends on the characteristics of these systems. For example, non-repudiation is not desirable in anonymous messaging systems designed to protect journalists and whistleblowers. Confidentiality is not required for public documents, but authenticity, integrity and even nonrepudiation typically are. Additionally, when accessing a website in which content is public, there is no need for the internet user to be authenticated.

2.1.2 Concrete security level and negligible functions

One of the main ideas of modern cryptography is to exploit difficult mathematical problems as a foundation to build cryptographic schemes that are hard to break. In general, for a problem to be used in cryptography, one has to be confident that it is hard to solve for the vast majority of its instances. Unfortunately however, proving lower bounds for the time complexity needed to solve certain computational problems remains one of the most important open problems in theoretical computer science.

The concrete security approach to define the hardness of breaking[6] a cryptographic scheme is described next. Suppose that the fastest known attacking algorithm takes 2^λ computing steps to solve the underlying problem of a given cryptographic scheme. Then we say that this cryptographic scheme provides λ bits of security.

[6] For now, we can consider an intuitive meaning for *breaking*, such as recovering the secret key or secret message of an user of a cryptographic scheme.

The main limitation of this approach is that the security level provided by some problem may need to be updated when better algorithms are discovered. This can be seen in practice in Chapter 4, which presents an attack against the binary variant or the permuted kernel problem that calls for a revision in the scheme's security level.

Another limitation is that, by the way it is defined, the security level depends on the computer architecture running the attack. This is circumvented by noticing that the gain by using more efficient architectures should be at most a small constant, and letting λ assume large values when instantiating schemes, such as 128, 192, or 256 bits.

In the proofs used in cryptography, we often use random processes and randomized algorithms, and it may be easier to talk about probability of success instead of amount of work to find a key or, usually, some more refined attacker objective. This motivates the use of λ as follows: an attacker who runs in polynomial time in λ should be able to succeed in their objective only with an extremely low probability, which should decrease fast as λ increases. This is formalized by means of negligible functions, that are defined next.

A function $\mu(\lambda) : \mathbb{N} \to \mathbb{R}$ is called negligible if, for every positive integer c, there exists an integer n_c such that

$$|\mu(\lambda)| < \frac{1}{\lambda^c}$$

for integers λ greater than n_c.

It is important to notice that, although the formal definition above is used when proving theoretical results, in practice, cryptographers often target the negligible function $2^{-\lambda}$ when considering the probability of attackers succeeding, or even when computing the probability of an operation, such as decryption, failing to complete correctly.

2.1.3 Symmetric-key cryptography

Although this dissertation does not deal directly with symmetric-key schemes, it is important for the reader to have at least an intuition on why these schemes are used so that they can better understand the function of a key encapsulation mechanism.

Symmetric-key cryptographic algorithms are those for which the same key is used for encryption and decryption. It is the oldest form of encryption and is very valuable for a number of applications such as encrypting drives and securing communication between two parties that hold the secret key. Examples of these types of schemes are the AES [DR99], which is still considered secure, and DES [Pub99], which is now obsolete.

The most important feature of symmetric-key algorithms is that they are much faster than public-key encryption. On the other hand, they present the following important limitation: in a setup where n users want to communicate among themselves, there are $\binom{n}{2} = n(n-1)/2$ keys that must be exchanged beforehand in a secure channel, such as face-to-face meeting. This is easier to handle with public-key schemes, where only n public keys are needed for the n users.

Other limitations of symmetric-key cryptography may be even more important depending on the application. For example, if key k is shared by Alice and Bob, there is no way to

prove who encrypted a message with key k. Furthermore, if k is compromised, every previous communication between Alice and Bob can be decrypted.[7] In Section 2.2.2, we can see how public-key and symmetric-key schemes can be combined to obtain the best of the two worlds: use public-key encryption to share a secret key to be used by fast symmetric-key schemes.

2.1.4 Hash functions and the random oracle model

In computer science, a general hash function is any function that takes an input of any size and outputs a fixed size string of bits. In cryptography, however, the definition is much more strict. A cryptographic hash function must satisfy two security properties: *preimage resistance* and *collision resistance*. Preimage resistance means that, given y, it is unfeasible to find x such that $H(x) = y$, for some hash function H. A hash function is collision resistant if, for any probabilistic polynomial-time attacker, it is unfeasible to output two different strings that share the same hash value. Notice that to break collision resistance is inherently easier than to break preimage resistance, since there is no target hash value, and any pair with the same hash will suffice.

Usually, it is required that a cryptographic hash function H whose output is ℓ bits long, formally $H : \{0,1\}^* \to \{0,1\}^\ell$, must provide ℓ bits of security against preimage computation to be used in the real world. That is, the best known algorithms for producing preimages for H must take around 2^ℓ operations. For finding collisions, the expected number of operations is lower in general as the collision-finding problem can be modeled by the *birthday problem* [KL21, Section A.4]. Therefore, the number of hashes we need to compute until we see a collision for H is about $\sqrt{2^\ell}$, corresponding to around $2^{\ell/2}$ operations, or $\ell/2$ bits of security against collision-finding attacks.

Modern hash functions are believed to be safe for against quantum computers. Similar to symmetric-key primitives, the best known quantum attacks against hash functions are based on Grover's [Gro96] search algorithm. Therefore, they can be made secure by using larger outputs, by doubling the output size, for example.

Constructions

There are two ways to instantiate hash functions. The most commonly used is by using constructions similar to symmetric-key schemes, which makes for very efficient hash functions. Some well known examples of hash functions following symmetric-key constructions are the SHA-2 [HE11] and Keccak [BDPA13], also known as SHA-3, and also older hashes that are not considered safe anymore, such as MD5 [Riv92] and SHA-1 [EJ01].

The other, which is called provably secure hash functions, consists in carefully defining the hash function in a way such that, if an attacker can find a collision, then a hard problem is solved. These are much less efficient, and therefore are not widely deployed. Some

[7] The security of previous communication against compromised keys is called *forward secrecy* and can be achieved by some public-key schemes.

examples are VSH [CLS06], which is based on the hardness of factoring, and FSB [AFS05], which is based on the hardness of the syndrome decoding problem.

Applications

When talking about hash functions, one is usually interested in the integrity and authenticity guarantees it can give, such as checksums and message authentication codes [BCK96]. This is a consequence of preimage resistance: given a hash value of a file, it is difficult for an attacker to produce a different file with the same hash.

Interestingly however, in the public-key setup, hash functions play a slightly different but very important role: they can guarantee that one operation is done before the other. In particular, if H is a cryptographic hash function and we let $b \leftarrow H(a)$, then it is secure to assume that a was generated before b. This simple observation is heavily exploited both when constructing signature schemes under what is called the Fiat-Shamir paradigm [FS86], and also to achieve the highest security notions for public-key encryption [FO99, HHK17a].

The Random Oracle model

In 1993, Bellare and Rogaway [BR93] questioned the gap between theory and practice of hash functions, and they take a more practical point of view to define what is expected by a hash function for secure cryptographic use. Instead of asking for what is the least possible security notion required for a hash function to be considered for cryptographic use, they ask: what are the properties that the hash functions used in the real-world really seem to have?

Take, for example, the Proof-of-Work required to validate a block B in some blockchain-based ledgers [GKL15]. The validity of such proofs come from the apparent hardness of finding a random number r such that $H(r, B) < t$, where t is some target number. However, this problem is not related to preimage or collision resistance in any way.

Bellare and Rogaway [BR93], then define a new model for hash functions called the Random Oracle model (ROM). Under this model, hash functions are assumed to behave as a Random Oracle, which is an idealize object that would require an exponential amount of memory to be instantiated. The behavior of a Random Oracle, which takes inputs $\mathbf{k}$ from $\{0,1\}^*$ and outputs $\mathbf{v}$ from $\{0,1\}^\ell$, is illustrated by Algorithm 2.1.

Algorithm 2.1 An idealized implementation of a Random Oracle.

```
1: procedure RandomOracle(k)
2:     if (k, v) ∈ T for some v then
3:         return v
4:     v ← Random sequence of ℓ bits
5:     Add (k, v) to T
6:     return v
```

Essentially, under the ROM, we can assume that the output of a hash function, is uniformly random, but anytime the same input is queried, even by different people, the

Random Oracle returns the same value. At this point, it is difficult to give a meaningful example of the importance of this kind of behavior when using hash functions together with other cryptographic primitives. But, as a high-level example, consider the case when a user wants to digitally sign a document D. The first step of every signing algorithm is to first compute the hash of the document (otherwise, if the document is too large, the computation would be impractical). If the outputs of the hash function are not uniformly random, it could be the case that $H(D)$ and $H(D')$ differ only by a small number of bits, considering D' as a minor corruption of D. Thus, depending on the signing algorithm, this could generate the same signatures for D and D'.

The ROM paradigm for proving the security of cryptographic constructions is then: first build a scheme, then prove its security under the ROM, and finally instantiate the random oracles with concrete hash functions that are believed to be safe. This approach has been very successful in building the most efficient cryptographic schemes and protocols. There are however, interesting caveats. Canetti, Goldreich, and Halevi [CGH98, CGH04] showed that there are cryptographic schemes that are secure under the ROM, but that are insecure when the random oracles are instantiated by any hash function. Although this result highlights important limitations of the ROM, their construction is rather artificial[8].

2.2 Public-key encryption

In this section, we begin by reviewing the definition of public-key encryption schemes and key encapsulation mechanisms. Then the security notions for these types of schemes are discussed. Notice that these concepts are essential for this dissertation: in Chapter 3, the HQC encryption scheme is attacked, while in Chapter 5, an improvement on BIKE key encapsulation mechanism is proposed.

2.2.1 Encryption schemes

A public-key encryption scheme is defined by a set of 4 algorithms: Setup, KeyGen, Encrypt and Decrypt. To instantiate an encryption scheme, one chooses a security level λ and runs the Setup algorithm. This algorithm, which typically involves a simple lookup in a parameters table, returns the public parameters of the scheme, that must be known and used for parties who want to communicate.

Now, each party runs the KeyGen algorithm to obtain a pair of keys: one is the secret key $\mathbf{k}_{\text{sec}}$ and the other is the public key $\mathbf{k}_{\text{pub}}$. The secret key must be stored as securely as possible, while the public key must be securely distributed. When defining encryption schemes and proving their security, we often assume that Alice and Bob have access to each other's public key. However, the secure distribution of public keys over the Internet is an important problem in the real world and its difficulty must not be overlooked.

[8]For example, the signing algorithm of the artificial signature scheme they constructed may output the secret key in some backdoor cases.

Suppose Alice wants to send a message to Bob. She then uses the ENCRYPT algorithm with her message and Bob's public key to obtain a ciphertext, which she sends to Bob. Then, Bob uses the DECRYPT algorithm together with his secret key to obtain Alice's original message.

It is often required that the probability of decryption failure is negligible on the security level. Formally, this means that

$$\Pr(\text{DECRYPT}(\mathbf{k}_{\text{sec}}, \text{ENCRYPT}(\mathbf{k}_{\text{pub}}, \mathbf{m})) \neq \mathbf{m}) \leq \mu(\lambda), \tag{2.1}$$

where μ is a negligible function, which is typically set as $\mu(\lambda) = 2^{-\lambda}$.

One of the main problems when designing a public-key encryption scheme is that, at the same time the public and secret keys, $\mathbf{k}_{\text{pub}}$ and $\mathbf{k}_{\text{sec}}$, are clearly related by a strong mathematical relationship, such as Equation 2.1, it must be unfeasible to recover $\mathbf{k}_{\text{sec}}$ from $\mathbf{k}_{\text{pub}}$.

Sometimes, the ENCRYPTION algorithm is presented as a randomized algorithm, and it is useful to make it deterministic by separating the randomness $\mathbf{r}$ used during encryption. We then use the notation $\text{ENCRYPT}(\mathbf{k}_{\text{pub}}, \mathbf{m}; \mathbf{r})$ to denote the deterministic encryption using the coins from $\mathbf{r}$.

2.2.2 Key encapsulation mechanisms

One of the drawbacks of public key cryptography is that all known schemes are much more computationally expensive than their symmetric-key counterparts. Therefore, public-key schemes are not suited to encrypt large messages. One solution then is to use public-key encryption schemes to encrypt a secret session key that will be used with a symmetric-key encryption scheme such as AES [DR99]. This is what a key encapsulation mechanism (KEM) is designed to achieve [CS03, Den03].

Similarly to encryption schemes, a KEM consists of four algorithms: SETUP, KEYGEN, ENCAPS, DECAPS. A secure public-key encryption scheme can be used to instantiate a KEM as follows. The SETUP and KEYGEN function are the same as the ones from the public-key encryption. The encapsulation function ENCAPS encapsulates a session key by first generating it as a random bit string, and then encrypting it using the ENCRYPT algorithm. The key decapsulation algorithm DECAPS is then equivalent to the DECRYPT algorithm. The session key is then used for symmetric-key encryption during the communication session.

2.2.3 Cryptographic security notions

When one gives a formal description of an encryption scheme, it is important to be very clear both with respect to what attacks it can resist and the power of an attacker. The three most common attack objectives for public-key cryptography are enumerated next.

1. Key recovery: given the public key, the adversary must recover the secret key.

2. Break one-wayness (OW): given a ciphertext and the public key, the attacker must recover the plaintext used to produce the ciphertext.

3. Break ciphertext indistinguishability (IND): given two fixed plaintexts $\mathbf{m}_0$ and $\mathbf{m}_1$, a public key $\mathbf{k}_{\text{pub}}$ and $\mathbf{c}$, the attacker must distinguish if $\mathbf{c}$ is the encryption of $\mathbf{m}_0$ or $\mathbf{m}_1$ under $\mathbf{k}_{\text{pub}}$.

Notice that key recovery is necessarily the hardest one: if one has the secret key, they can easily recover any plaintext message and distinguish ciphertexts by performing decryption operations. Furthermore, notice that, even though it may seem easy to build a cipher resistant to distinguishing attacks, no scheme with a deterministic encryption algorithm[9] provides ciphertext indistinguishability: the attacker can just encrypt $\mathbf{m}_0$ and $\mathbf{m}_1$ with the public key $\mathbf{k}_{\text{pub}}$ and compare the results with $\mathbf{c}$.

The two most important types of attacks, with respect to the interaction it can make with the holder of the secret key are the following:

1. Chosen-plaintext attack (CPA): the attacker can choose a number of plaintext and see their associated ciphertexts.

2. Chosen-ciphertext attack (CCA): the attacker can choose ciphertexts and ask to see the decryption of these ciphertexts.

Security notions are then formed by combining one attacker goal with the type of attack. For example: OW-CPA is a scheme that is one-way under chosen-plaintext attacks, while IND-CCA is a scheme whose ciphertexts are indistinguishable under chosen-ciphertext attacks.

In general, public-key encryption algorithms are presented in the OW-CPA format, because it is the easiest and cleanest way to understand the underlying mathematics. In practice however, most applications require high security guarantees such as IND-CPA or even IND-CCA. Luckily, there are generic conversion algorithms that take OW-CPA schemes and transform them into IND-CPA and IND-CCA schemes, which are described next.

2.2.4 Security conversions

In this section, we discuss the techniques one can use to increase the security of a public-key encryption scheme. In other words, given any scheme achieving some security notion, how can we transform it to achieve a higher security notion? We are mainly interested in transformations that take an OW-CPA public-key encryption scheme to build an IND-CCA secure scheme.

These transformations are often involved and therefore only an intuitive explanation of them is provided here. The main takeaway is that we can study KEMs or PKEs using

[9] This includes most textbook variants of RSA and Elliptic Curves, for example.

their pure, that is OW-CPA, description, if we keep in mind that they must be converted to a higher security when used in practice.

First let us recall what the IND-CCA security notion tries to capture. Intuitively, the most important requirement is that an attacker should not gain any useful information on a ciphertext from observing the decryption of other ciphertexts. The way most of the IND-CCA security conversions work is by making it nearly impossible to generate a new ciphertext without knowing the associated plaintext. Furthermore, notice that even the construction of new ciphertexts from other ciphertexts should be difficult. In cryptography research, these conditions are encompassed by the definitions of plaintext-awareness and non-malleability.

In 1999, Fujisaki and Okamoto [FO99] showed a generic transformation, denoted FO, that takes and OW-CPA public-key encryption scheme and build an IND-CCA secure scheme. Their construction is proven secure under the Random Oracle model and it uses a symmetric-key scheme and 2 hash functions. More recently, there have been some alternative proposals such as the one by Hofheinz, Hövelmanns and Kiltz [HHK17a]. These achieve the same objective as the original FO transformation, but do not require a symmetric-key scheme and are more suited for post-quantum encryption schemes, for which there is usually some negligible decryption failure probability.

Fujisaki-Okamoto transformation with implicit rejection

Of the transformations proposed by Hofheinz, Hövelmanns and Kiltz [HHK17a], one of the most commonly used is called the Fujisaki-Okamoto with implicit rejection. This is one of the strongest conversions presented by the authors as, to be transformed into an IND-CCA secure scheme, a given scheme needs only to be OW-CPA and have negligible decryption failure rate.

Let PKE = (Setup, KeyGen, Encrypt, Decrypt) be some OW-CPA public key encryption scheme with negligible decryption failure rate. The Fujisaki-Okamoto with implicit rejection that takes PKE and produces an IND-CCA secure KEM is shown as Algorithm 2.2 using two auxiliary hash functions $\mathcal{H}$ and $\mathcal{G}$ that are assumed to behave like Random Oracles. Notice that the Setup is exactly the same for both. Each step of this transformation is analyzed in more detail in the next paragraphs.

The KemKeyGen is almost exactly as the PKE's KeyGen algorithm, except that it generates a random secret seed σ, which will be part of the new secret key. This secret seed is used only when deriving a fake key when implicit rejection is required by the decapsulation algorithm. One important thing to notice is that we denote a sequence of λ ones as 1^λ, which is passed to the KeyGen algorithm. This is a technical condition[10] to allow for KeyGen to run in time polynomial in the length of 1^λ, which is λ.

Now consider the Encapsulation procedure. Notice that the input is only the public key $\mathbf{k}_{\text{pub}}$ of the intended receiver, because there is no message: the encapsulation generates

[10]This is required from the complexity theory notion of what makes an algorithm be probabilistic polynomial-time, but in practice we just pass λ directly to the functions.

Algorithm 2.2 The Fujisaki-Okamoto transformation with implicit rejection.

procedure KEMKEYGEN(1^λ)
  $\mathbf{k}_{\text{pub}}^{\text{PKE}}, \mathbf{k}_{\text{sec}}^{\text{PKE}} \leftarrow \text{KEYGEN}(1^\lambda)$
  $\sigma \leftarrow$ Random sequence of λ bits
  $\mathbf{k}_{\text{pub}} \leftarrow \mathbf{k}_{\text{pub}}^{\text{PKE}}$
  $\mathbf{k}_{\text{sec}} \leftarrow \left(\mathbf{k}_{\text{sec}}^{\text{PKE}}, \sigma\right)$
  return $\mathbf{k}_{\text{sec}}, \mathbf{k}_{\text{pub}}$

procedure ENCAPSULATE($\mathbf{k}_{\text{pub}}$)
  $\mathbf{m} \leftarrow$ Random element in the message space of PKE
  $\mathbf{r} \leftarrow \mathcal{G}(\mathbf{m}, \mathbf{k}_{\text{pub}})$ ▷ The randomness to be used by ENCRYPT
  $\mathbf{c} \leftarrow \text{ENCRYPT}(\mathbf{k}_{\text{pub}}, \mathbf{m}; \mathbf{r})$
  $\mathbf{k} \leftarrow \mathcal{H}(\mathbf{m}, \mathbf{c})$ ▷ The encapsulated key to be shared
  return $\mathbf{c}, \mathbf{k}$

procedure DECAPSULATE($\mathbf{k}_{\text{sec}}, \mathbf{c}$)
  Parse $\mathbf{k}_{\text{sec}}^{\text{PKE}}, \sigma \leftarrow \mathbf{k}_{\text{sec}}$
  $\mathbf{k}_{\text{reject}} = \mathcal{H}(\sigma, \mathbf{c})$ ▷ The fake key used for implicit rejection
  $\hat{\mathbf{m}} \leftarrow \text{DECRYPT}(\mathbf{k}_{\text{sec}}^{\text{PKE}}, \mathbf{c})$
  if $\hat{\mathbf{m}} = \perp$ **then**
    return $\mathbf{k}_{\text{reject}}$
  $\hat{\mathbf{r}} \leftarrow \mathcal{G}(\hat{\mathbf{m}}, \mathbf{k}_{\text{pub}})$
  $\hat{\mathbf{c}} \leftarrow \text{ENCRYPT}(\mathbf{k}_{\text{pub}}, \hat{\mathbf{m}}; \hat{\mathbf{r}})$ ▷ Reencryption
  if $\hat{\mathbf{c}} \neq \mathbf{c}$ **then**
    return $\mathbf{k}_{\text{reject}}$
  $\mathbf{k} \leftarrow \mathcal{H}(\hat{\mathbf{m}}, \mathbf{c})$ ▷ The shared decapsulated key
  return $\mathbf{k}$

a fresh random key to be used with a symmetric-key algorithm. The first thing is to generate a random seed $\mathbf{m}$, that is what will be encrypted using PKE. Notice, however, that $\mathbf{m}$ will be encrypted using randomness $\mathbf{r}$ that comes from the hash of $\mathcal{G}(\mathbf{m}, \mathbf{k}_{\text{pub}})$, resulting in ciphertext $\mathbf{c}$. This is one critical step of the transformation: since the randomness depend on $\mathbf{m}$, if an adversary alters the ciphertext, the intended receiver can detect this corruption by reencrypting the message and comparing the result with the ciphertext. Furthermore, since $\mathbf{r}$ also depends on $\mathbf{k}_{\text{pub}}$, this makes it difficult for multi-target search for colliding values of $\mathbf{r}$. The symmetric key to be shared is $\mathbf{k} = \mathcal{H}(\mathbf{m}, \mathbf{c})$.

The decapsulation is the most complex step. It starts by computing the implicit rejection key, which is a fake key that is used when it detects some problem, that can be a decryption failure or a detection of malformed ciphertext. It is called implicit because the receiver does not immediately tells the sender that there was a problem. This makes it more difficult for an attacker mounting a CCA attack to distinguish whether his manipulation of the ciphertext resulted in a valid ciphertext or not, and it is important for the proof technique used by Hofheinz, Hövelmanns and Kiltz [HHK17a].

Then, the decapsulation procedure calls the decryption procedure of PKE to recover the

base seed $\mathbf{m}$. If there is a decryption failure, return the implicit rejection key, as discussed below. Otherwise, engage in a reencryption. The necessity of reencryption is that, although the decryption was successful, there is not yet any guarantees that the ciphertext was not manipulated. However, if we encrypt $\mathbf{m}$ again and obtain a different value of $\mathbf{c}$, then the ciphertext was altered and we use the implicit rejection key. Otherwise, we consider that everything went fine and we can use the shared symmetric key $\mathbf{k} = \mathcal{H}(\hat{\mathbf{m}}, \mathbf{c})$, which will hopefully be the same key as $\mathcal{H}(\mathbf{m}, \mathbf{c})$.

2.3 Digital signatures

Chapter 4 presents an attack against the Permuted Kernel Problem (PKP), which is the fundamental problem associated with a signature scheme. The attack is presented in a way that does not require the reader to understand digital signatures, since it consists of a series of binary linear algebra algorithms. However, it is useful for the reader to have at least some background on digital signatures, so that they can better understand how PKP is connected to cryptography.

2.3.1 Definition of signature schemes and their security

A signature scheme consists of 4 algorithms: Setup, KeyGen, Sign and Verify. The behavior of the Setup and KeyGen are analogous to encryption schemes. The Setup amounts for searching in a table for the public parameters achieving a given security level, while KeyGen generates a pair of public and secret keys. The Sign algorithm is given a message $\mathbf{m}$ and a secret key $\mathbf{k}_{\text{sec}}$, and produces a signature σ. The Verify algorithm is given $\mathbf{m}$, a public key $\mathbf{k}_{\text{pub}}$ and the candidate signature σ, and decides whether σ is a valid signature of $\mathbf{m}$ under $\mathbf{k}_{\text{pub}}$ or not.

Desired security notion for signature schemes

Similarly to encryption schemes, there are a number of security notions for signature schemes and also security conversions that can transform schemes to achieve higher security notions, usually with some performance penalties. The most widely required security notion for digital signature schemes is called existentially unforgeability under chosen message attacks (EU-CMA), which is explained next. Suppose an attacker, who is attacking Alice, can ask Alice for valid signatures for any message of their choosing. The signature scheme is considered EU-CMA if, even with this powerful attacking setup, the attacker cannot build a valid signature for a message that was not previously signed by Alice herself.

Constructing signature schemes

There are three paradigms to construct a digital signature: the hash-and-sign paradigm, using the Fiat-Shamir transformation, and signatures based on hash functions only. Arguably, the most well known among them is the hash-and-sign approach [BR96], which

works as follows. Suppose the signer knows a secret information, called the trapdoor, that allows them, and only them, to compute a preimage x of a public function f given any[11] $y = f(x)$. Under the hash-and-sign approach, the signer would first hash a message m to the image of f, obtaining $y = \textsc{Hash}(m)$, and then use the trapdoor information to compute some x such that $y = f(x)$. Since f is public, anyone can verify that x is a valid signature of m by first hashing m and comparing the result with $f(x)$.

Hash-based signatures are an interesting type of signature whose security is completely based on the security of the hash functions used to instantiate it. The basic idea comes from a report Lamport [Lam79] published in 1979 and is described next. Let H and G be hash functions, the secret key is a seed s and the public key is a sequence of n hashes of the form

$$\mathbf{k}_{\text{pub}} = \big((\mathbf{y}_1^0, \mathbf{y}_1^1), \ldots, (\mathbf{y}_n^0, \mathbf{y}_n^1)\big),$$

where each $\mathbf{y}_i^b = G(H(s, b, i))$. The signature σ of an n-bit message $\mathbf{m} = (m_1, \ldots, m_n)$ is the sequence of preimages, with respect to G, of the bits in each position, which corresponds to $\sigma = (H(s, m_1, 1), \ldots, H(s, m_n, n))$. One very important limitation of this signature is that public key $\mathbf{k}_{\text{pub}}$ can only be used one time – each time $\mathbf{k}_{\text{pub}}$ is reused, an attacker can combine the preimages learned to craft and sign new messages. However, this idea can be improved to allow for the same signing key to be reused safely but with an increased signing time, as is the case for SPHINCS+ [ABB+19].

The Fiat-Shamir [FS86] paradigm is introduced in the next section for two reasons. The first is that it is difficult to explain it in one paragraph since it requires some discussion on zero-knowledge proofs. The second is that PKP-based signatures are based on this paradigm, which makes it important to explain the Fiat-Shamir paradigm in a little more detail.

2.3.2 The Fiat-Shamir paradigm

The Fiat-Shamir [FS86] paradigm uses interactive proofs of knowledge to build signatures that are secure under the Random Oracle model and assuming the underlying mathematical problem is intractable. A proof of knowledge, which is a type of zero-knowledge proof [GMR19], is a powerful proving technique in which two parties, a prover and a verifier, interact with the following goal: the honest prover, who knows a solution to a problem, wants to prove to the verifier that they know a solution to a problem, but without revealing any additional information on the solution. For this to be possible, the verifier needs to tolerate some negligible probability of being falsely convinced that the prover really knows the solution.

One interesting example of zero-knowledge proof is what is called the Schnorr's identification scheme [Sch89]. In this scheme, the prover wants to convince the verifier that

[11] The *any* qualifier is very important here. If that is not the case, the hash function could generate values y for which the preimage could not be computed and a signature scheme wouldn't be possible to construct.

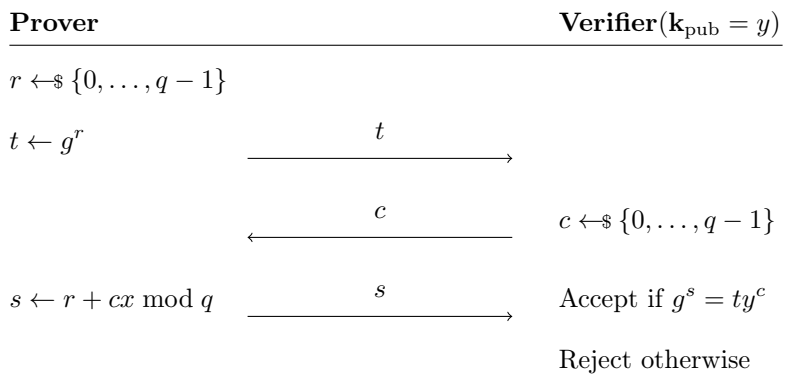

Figure 2.1: Schnorr's identification scheme: an efficient zero-knowledge proof that Prover knows the discrete logarithm x such that $y = g^x$. The symbol $\leftarrow\!\!\$$ denotes an uniform selection from a given set.

he knows a solution for an instance of the discrete logarithm problem. In this setup, both prover and verifier agree on a cyclic group G of order q and one of its generator g, This means that $G = \left\{g^i : i \in \{0, \ldots, q-1\}\right\}$. Suppose that they both know a value $y \in G$ but the prover claims that he knows an x such that $y = g^x$, that is, he wants to prove knowledge of a discrete logarithm, a problem which, when defined in certain groups, is believed to be very hard for classical computers to solve. The Schnorr [Sch89] protocol for proof of knowledge is shown in Figure 2.1.

Let us see why this is a valid zero-knowledge proof. If the prover is honest, then the verifier always accepts since $g^s = g^{r+cx} = g^r g^{cx} = t(g^x)^c = ty^c$. Now let us see what can be learned by x from this interaction. Clearly c does not contain any information on x since it was randomly chosen by the verifier. Now, notice that s, even considered together with c, also do not carry any information on x, since s, without knowing r, is simply a random variable in $\{0, \ldots, q-1\}$. Finally, the value of t is completely determined from s and c using the equation $t = g^s y^{-c}$. Therefore, anyone seeing the interaction (t, c, s) learns nothing about x. The question is then: why can we be so sure that the prover knows x?

Since anyone can forge a valid interaction transcript (t, c, s) by fixing first s and c before t, the answer is on the prover's availability in first committing to t and then answering the challenge c. If the prover is honest, then, after committing to t, the he must be able to give valid answer s_1 and s_2 to at least two different challenges c_1 and c_2. If this is the case, then the prover must know x, since $\frac{s_1 - s_2}{c_1 - c_2} = \frac{(c_1 - c_2)x}{c_1 - c_2} = x$. And, if this is not the case, he only knows how to answer one challenge and will be caught cheating with overwhelming probability $1 - 1/q$.

Now, since to prove knowledge of x, the challenge must be generated after the commitment t, this proof can be made non-interactive if we let the challenge $c = H(t)$ for some cryptographic hash function H. Furthermore, we can even transform this non-interactive proof of knowledge into a signature scheme if we let the challenge depend also on the message to be signed, by setting $c = H(t, m)$. This is precisely how the Fiat-Shamir paradigm

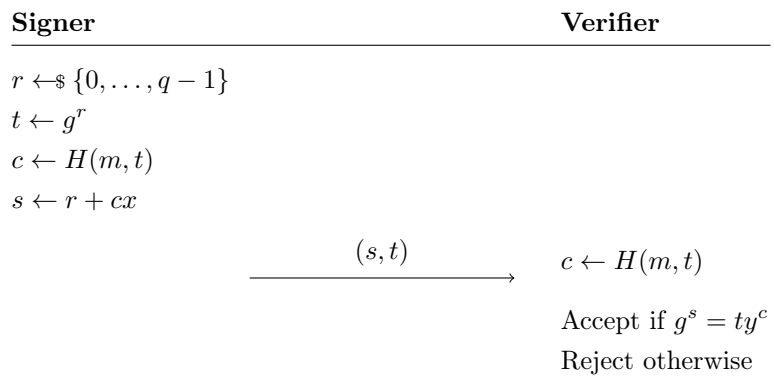

Figure 2.2: Schnorr's signature scheme obtained by applying the Fiat-Shamir transform over Schnorr's identification scheme. The signer's secret key is $\mathbf{k}_{\text{sec}} = x$ and the public key is $\mathbf{k}_{\text{pub}} = y = g^x$. The symbol $\leftarrow\$$ denotes an uniform selection from a given set.

is used to construct signatures. This is formalized in Figure 2.2.

One interesting thing about these Fiat-Shamir constructions is that we can build signature schemes over really hard problems, since, under weaker assumptions than the ROM, zero-knowledge proofs exist for all problems in NP [GMW91]. However, they are not always as efficient as Schnorr's scheme, and may require a lot of interactions for the verifier to get a negligible probability of being fooled. PKP-DSS [BFK+19] and Binary PKP [LP12], which are the focus of Chapter 4, are representatives of this case: they are both constructed over an NP-hard problem called the permuted kernel problem.

2.4 Secure cryptography implementation

The mathematical security models described in the previous sections are very useful when designing cryptographic schemes. The success of these models comes from forcing designers of schemes to be very specific, which helps researchers identify what are the weak points that can be attacked. However, for a scheme to be useful in practice, the theoretical hardness is not enough: it must also have an efficient and secure implementation.

The implementation of cryptographic algorithms can be very challenging because there are a great number of possible sources of leakages. In this section, we start by briefly reviewing the main sources of extra information attackers can use to break a cryptographic implementation. Then we describe the main techniques used to avoid timing-based sources of leakages, which is arguably the most critical leakage an implementation must avoid for it to be used in the real world.

2.4.1 Side-channel attacks

Side-channel attacks exploit both implementation aspects of a cryptographic scheme and the architecture under which the scheme is deployed. The most common types of side-channel attacks exploit the following:

- timing information [Koc96],
- patterns of memory access [YGH17, ASK07]
- power consumption and electromagnetic (EM) emissions [KJJ99].

Of these, the most dangerous leakages are the first two, since they can be used to mount remote attacks [BB05], without access to the physical device in which a cryptographic operation is deployed. Luckily, these two are the easiest to mitigate using a paradigm known as constant-time programming.[12] These are implementations whose execution time, memory access and branching do not depend on secret inputs. The techniques to mitigate power-based and EM-based side-channels are usually more involved, and we consider them to be outside the scope of this dissertation.

In their post-quantum standardization process call for submissions [oST16], NIST explicitly states their preference for schemes that can be made resistant to side-channel attacks with the lowest overhead. As such, most implementations submitted to NIST, at least in the third round, come with an optimized constant-time implementation.

2.4.2 Constant-time programming

There are 3 main principles that must be followed under the constant-time programming paradigm [JFB+22]:

- use only constant-time arithmetic operations on secret inputs,
- no branches should depend on secret inputs, and
- memory access patterns should not depend on secrets.

These may not appear to be hard to enforce, and in fact, there are even some tools for automatic verification that a given program meets these conditions [Lan10]. However, there are a number of aspects that make constant-time implementations challenging. Of those, two that are particularly important are discussed next.

The first problem is that sometimes we need to use very different algorithms than those that are used in the mathematical specification. Some algorithms are not only easier to make constant-time than others, but yield more efficient procedures. Consider the problem of sorting. It is easy to see that the pattern in which quicksort accesses elements of the array is very dependent on the entries of the array. Therefore, when sorting is required, there is usually no easy solution and the developer needs to understand the parameters of the problem to choose among different options. For example, if the number of elements is small, maybe a constant-time bubble sort can be used while if there are not many different elements, a constant-time counting sort may be better. In more complex cases,

[12] Note that *constant-time* does not mean that the execution time is constant, because there are a number of other variables that affect the CPU execution time. Some authors suggest the term *secret-independent time* instead of *constant-time* but this is less common.

there are more complex sorting-network approach which makes for a generic constant-time algorithm [Ber19].

The second problem is that, even if all the 3 principles are followed by the programmer in a high-level language, the compiler may transform their code into a non-constant-time executable. To solve this problem, it is important to look into the assembly code generated by the compiler to ensure that it did not inserted timing vulnerabilities. Recently, a new language [ABB+17a] was proposed, which is designed specifically for cryptography, and for which the compiler does not insert timing vulnerabilities. However, most of cryptography code is still written in C.

Maybe because of the high dependency on the architecture and also on the compiler, there are few resources on constant-time programming that are friendly to newcomers. One exception, however, is the short course by Hernández et al. [HCAL15]. Another very useful resource is Pornin's discussion on the constant-time implementation of BearSSL [Por18]. In the next 3 sections, we provide a more detailed explanations of how to apply the 3 principles of constant-time programming.

The arithmetic operations on secrets should be constant-time

In this work, we assume that the execution time of the following operations are independent of their operands.

- The bitwise operations over unsigned integers: *and*, *or*, *xor*, and *not*, denoted by `&`, `|`, `^` and `~`, respectively.
- Sum and subtraction of unsigned integers.
- Left and right shifts of unsigned integers, denoted by `<<` and `>>`, respectively.

These assumptions are consistent with most cryptography implementations in modern processors. However, notice that left and right shifts may not be constant-time in processors that does not come with a barrel shifter, which may not be the case for some old processors.[13] Notice that the multiplication of unsigned integers is sometimes assumed to be constant-time in modern processors, but we do not need this assumption in this work for the constant-time implementation we describe in Chapter 5. Notably, division and modulo operations are well known to be problematic under a number of architectures, and, as such, we avoid these instructions in our implementation. When these operations are strictly needed, we implement them in constant-time using other arithmetical operations[14].

Avoiding branches dependent on secret data

One common source of information leakage in cryptography implementation comes from branching. In cases when the branch selection depends on bits of the secret key, this

[13] Such as Intel's Pentium IV under the NetBurst microarchitecture [Por18].

[14] This is done for the modulo operation needed for the sampling required by the constant-time implementation of FixFlip, which is discussed in Section 5.7.

can lead to key-recovery attacks when the attacker can measure the time taken by the operation. Similarly, if the branch is selected depending on bits of the secret message, this can lead to message recovery attacks.

The main technique to avoid branching is to perform the computation in both branches, and then select the desired result by using condition masks. This is exemplified in the snippet below. There are two implementations of the same procedure that conditionally selects value `a` or value `b` depending on the parity of the integer `parity_condition`. The safe implementation is constant-time and we can see the usage of the condition variable `mask` being used to select the right result.

```
#include <stdint.h>

uint32_t unsafe_conditional_select(uint32_t a, uint32_t b, uint32_t
    parity_condition) {
  uint32_t bit = parity_condition % 2;
  if (bit == 0)
    return a;
  else
    return b;
}

uint32_t safe_conditional_select(uint32_t a, uint32_t b, uint32_t
    parity_condition) {
  uint32_t bit = parity_condition & 1;
  uint32_t mask = -bit;
  // Notice that we have two cases
  //   mask = 0x00000000 if bit = 0;
  //   mask = 0xFFFFFFFF if bit = 1;
  return (b & mask) | (a & ~mask);
}
```

The pattern of memory access should not depend on secrets

The last rule is that one should not use secret indexes to access data in memory directly. This is very important since they can result in cache-timing attacks, where the attacker gets information on the secret indexes by counting cache misses or hits [ASK07]. This type of attack was shown to be devastating when combined with the Flush+Reload [YGH17, ASK07] method, where the attacker can flush selected cache lines from L3 and then, by timing load operations, determine whether these cache lines were used by the victim's process.

When we need to access a secret index of an array, the countermeasure is then to touch every element of the array, but only make the desired modification based on condition masks. The example below shows how to use this idea to safely select an entry from a table whose index is secret.

```
#include <stdint.h>
```

```
2
3 #define TABLE_SIZE 4
4
5 uint32_t Table[TABLE_SIZE] = {0xF0, 0xF1, 0xF2, 0xF3};
6
7 uint32_t unsafe_table_select(uint32_t secret_index) {
8     return Table[secret_index];
9 }
10
11 uint32_t safe_table_select(uint32_t secret_index) {
12     uint32_t value = 0;
13     for (int i = 0; i < TABLE_SIZE; i++) {
14         uint32_t mask = -(i == secret_index);
15         value |= Table[i] & mask;
16     }
17     return value;
18 }
```

This makes it nontrivial for cryptographic algorithms to sort or shuffle an array in constant-time. These routines usually need to be optimized separately for each case, by taking into account the specific parameters used by the scheme.

2.5 Coding theory and applications to cryptography

Coding theory [Sha48, vT93, Rom92] is an important subject with several real-world applications. Two of its most common applications are compressing large files and encoding messages for reliable transmission. While these appear to have no connection with cryptography, one of the oldest public-key encryption schemes, namely the McEliece scheme [McE78], is based on the hardness of some coding theory problems.

In coding theory, messages are defined as a sequence of symbols, and the sender may use what is called a *code* to add redundancy to this message, so that it can be reliably transmitted through a potentially noisy channel [Sha48]. The main problem is then to devise good codes, that are both efficient and can deal with the noise patterns that are caused by target channel[15].

Figure 2.3 illustrates the coding theory model for message transmission through noisy channels. First the source (or sender) chooses a message $\mathbf{m}$ from a public and fixed set $\mathcal{M}$ of possible messages. It then passes the message $\mathbf{m}$ through an encoder, that adds redundancy to the message, obtaining $\mathbf{c}$. When $\mathbf{c}$ is transmitted, the channel may add some noise $\mathbf{e}$ to it, resulting in $\mathbf{c}'$. The recipient then applies the decoding algorithm to extract the errors from $\mathbf{c}'$, obtaining $\hat{\mathbf{m}}$. If the recipient is lucky enough so that $\mathbf{c}'$ is sufficiently similar to $\mathbf{c}$, then, with high probability $\hat{\mathbf{m}}$ should be equal to $\mathbf{m}$.

[15] Notice that different transmission channels have different noise characteristics, for example the noise caused by a scratch in compact disc (CD) is different from that in a wireless transmission.

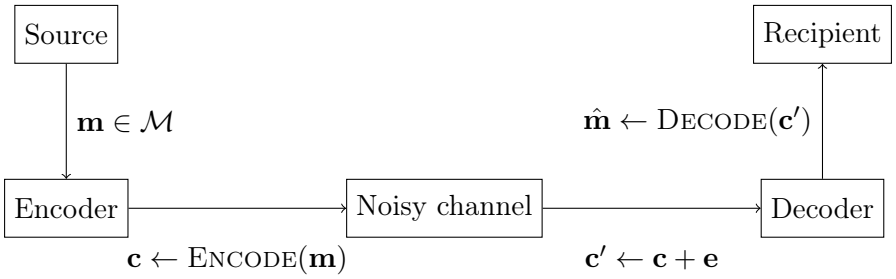

Figure 2.3: Transmission of a message $\mathbf{m}$ through a noisy channel.

2.5.1 Binary linear codes

Consider the simplest case when messages consist of sequences of k bits, that is, each symbols is an element of the binary field $\mathbb{F}_2$. Formally, the message space is then $\mathcal{M} = \mathbb{F}_2^k$. While the encoding function may, in theory, be any injective function, for it to be efficient, it must have a compact description. Arguably, the easiest solution for this compact representation is to use a linear function for encoding, which is easily achieved using matrix-vector multiplications. This motivates the definition of a binary linear code, which are one of the most well-studied types of codes.

Definition 2.5.1 (Binary linear codes and codewords)**.** A *binary* $[n, k]$*-linear code* is a k-dimensional linear subspace of $\mathbb{F}_2^n$, where $\mathbb{F}_2$ denotes the binary field. Elements of a given linear code are called its *codewords*.

Since they are vector spaces, binary linear codes have compact representations: they can be fully described as the image or the kernel of binary matrices. These two types of representation motivates the concepts of generator and parity-check matrices below.

Definition 2.5.2 (Generator and parity-check matrices)**.** Let $\mathcal{C}$ be a binary $[n, k]$-linear code. If $\mathcal{C}$ is the linear subspace spanned by the rows of a matrix $\mathbf{G}$ of $\mathbb{F}_2^{k\times n}$, we say that $\mathbf{G}$ is a *generator matrix* of $\mathcal{C}$. Similarly, if $\mathcal{C}$ is the kernel of a matrix $\mathbf{H}$ of $\mathbb{F}_2^{(n-k)\times n}$, we say that $\mathbf{H}$ is a *parity-check matrix* of $\mathcal{C}$.

Then, if $\mathbf{G}$ e $\mathbf{H}$ are any pair of generator and parity-check matrices of a given binary $[n, k]$-linear code $\mathcal{C}$, then

$$\mathcal{C} = \{\mathbf{mG} : \mathbf{m} \in \mathbb{F}_2^k\} = \{\mathbf{c} \in \mathbb{F}_2^n : \mathbf{cH}^T = 0\}.$$

The denomination *parity-check matrix* comes from the fact that, given one such matrix $\mathbf{H} = (h)_{ij}$, for a vector $\mathbf{c} = [c_1 \dots c_n]$ to be in the code represented by $\mathbf{H}$, the following set of equations must be satisfied

$$\begin{array}{ccccccccc}
c_1h_{11} & + & c_2h_{12} & + & \dots & + & c_nh_{1n} & = & 0, \\
c_1h_{21} & + & c_2h_{22} & + & \dots & + & c_nh_{2n} & = & 0, \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\
c_1h_{r1} & + & c_2h_{r2} & + & \dots & + & c_nh_{rn} & = & 0.
\end{array}$$

Since the operations are carried over $\mathbb{F}_2$, these are said to be parity check equations.

When decoding a corrupted codeword, it is usually important to know which parity check equations are not satisfied. This information, which is commonly used by decoders to correct possible errors, is called *syndrome* and is formally defined bellow.

Definition 2.5.3 (Syndrome)**.** The syndrome of a vector $\mathbf{v}$ of $\mathbb{F}_2^n$, with respect to a parity-check matrix $\mathbf{H}$ is the vector $\mathbf{v}\mathbf{H}^T$. Notice that $\mathbf{v}\mathbf{H}^T = \mathbf{0}$ if, and only if, vector $\mathbf{v}$ is in the code represented by $\mathbf{H}$.

Additionally, two important concepts for analyzing codes are (Hamming) *weight* and (Hamming) *distance*, which are defined next.

Definition 2.5.4 (Weight)**.** The Hamming *weight* of a vector $\mathbf{v}$, denoted by $\mathrm{w}(\mathbf{v})$, is the number of its non-null entries.

Definition 2.5.5 (Distance)**.** The Hamming *distance* between two vectors of the same length $\mathbf{u}$ and $\mathbf{v}$ of $\mathbb{F}_2^n$ is denoted by $\mathrm{d}(\mathbf{u}, \mathbf{v})$, and consists of the number of coordinates in which they differ. Notice that, in the binary case, it holds that $\mathrm{d}(\mathbf{u}, \mathbf{v}) = \mathrm{w}(\mathbf{u} + \mathbf{v})$.

2.5.2 Hard problems and applications to cryptography

Coding theory provides two hard problems that can be used in cryptography: the syndrome decoding problem and the problem of finding codewords of small weight. Both problems are known to be NP-hard [BMVT78], and they are defined as follows.

Definition 2.5.6 (Syndrome decoding problem)**.** Consider the following input: a binary matrix $\mathbf{H} \in \mathbb{F}_2^{k \times n}$, a vector $\mathbf{s} \in \mathbb{F}_2^k$, and an integer $w > 0$. The syndrome decoding problem asks for a $\mathbf{v}$ of weight w such that $\mathbf{v}\mathbf{H}^T = \mathbf{s}$.

Definition 2.5.7 (Finding codewords of small weight)**.** Consider the following input: a binary matrix $\mathbf{H} \in \mathbb{F}_2^{k \times n}$ and an integer $w > 0$. The problem is to find a codeword $\mathbf{c}$ such that $\mathbf{c}\mathbf{H}^T = \mathbf{0}$ and $\mathrm{w}(\mathbf{c}) \leq w$.

In 1978, McEliece [McE78] showed how to use the intractability of these problems to build a public-key encryption scheme. His rather elegant idea works as follows. For the key generation process, one picks at random a binary linear Goppa code [Gop70, Ber73]. The private key is a representation of this code that allows for efficient decoding, while the public key is a scrambled generator matrix of the Goppa code. When Alice wants to encrypt a message to Bob, she first uses Bob's public generator matrix to encode the message and then she intentionally adds random errors to the encoded message. Since only Bob knows the efficient decoder, only he can correct the errors and obtain the original message. This proposal was later improved by Niederreiter [Nie86].

Notice that it is critical for the security of the scheme that the secret code representation, which gives the efficient decoder, is not recoverable from the public representation. Even though Goppa codes are still believed to safely instantiate the McEliece

scheme [BCL+19], the public keys are too large for some applications. Other more efficient proposals for different code families rely on a quasi-cyclic or quasi-dyadic structure for compact representation [Gab05, BCGO09, BCGM07, MB09]. Although they obtain compact keys, most of them were shown insecure [OTD10, FOPT10, FOP+16]. A noticeable exception is the use of quasi-cyclic moderate-density parity-check codes [MTSB13, ABB+21], which appears to be resistant against cryptanalysis and also yields small keys.

In 2003, Alekhnovich [Ale03] proposed a new code-based scheme whose security relies purely on the decoding problem of random linear codes. That is, different from the McEliece scheme [McE78], Alekhnovich's scheme does not require a secret representation of the code. The problem however, is that the keys are much larger, making the scheme far from practical. To make it more efficient, Aguilar-Melchor et al. [AMBD+18] proposed the use of quasi-cyclic codes as a variant of Alekhnovich's, and also discussed additional techniques to achieve negligible decryption failure probability. Their construction was refined and gave origin to HQC [MAB+18], a code-based candidate in NIST's PQC process that moved to the 4th round.

Chapter 3

A timing attack on the HQC encryption scheme

Abstract. The HQC public-key encryption scheme is a promising code-based submission to NIST's post-quantum cryptography standardization process. The scheme is based on the decisional decoding problem for random quasi-cyclic codes. One problem of the HQC's reference implementation submitted to NIST in the first round of the standardization process is that the decryption operation is not constant-time. In particular, the decryption time depends on the number of errors decoded by a BCH decoder. We use this to present the first timing attack against HQC. The attack is practical, requiring the attacker to record the decryption time of around 400 million ciphertexts for a set of HQC parameters corresponding to 128 bits of security. This makes the use of constant-time decoders mandatory for the scheme to be considered secure.
Keywords: HQC, post-quantum cryptography, timing attack, BCH decoding

3.1 Introduction

Hamming Quasi-Cyclic (HQC) [MAB+18] is a code-based public-key encryption scheme. It is based on the hardness of the quasi-cyclic syndrome decoding problem, a conjectured hard problem from Coding Theory. It offers reasonably good parameters, with better key sizes than the classical McEliece scheme [McE78, BCL+19, ACP+18], but without relying on codes with a secret sparse structure, such as QC-MDPC [MTSB13] and QC-LDPC [Bal14].

One of the most interesting features HQC provides is a detailed analysis of the decryption failure probability, which makes it possible to choose parameters that provably avoid reaction attacks [GJS16, FHS+17] that compromise the security of QC-LDPC and QC-MDPC encryption schemes. This makes it one of the most promising code-based candidates in NIST's Post-Quantum standardization process. However, the negligible probability of decoding failure comes at the expense of low encryption rates.

The scheme uses an error correction code $\mathcal{C}$ as a public parameter. The secret key is

a sparse vector, while the public key is its syndrome with respect to a systematic quasi-cyclic matrix chosen at random, together with the description of this matrix. To encrypt a message, the sender first encodes it with respect to the public code $\mathcal{C}$, then adds to it a binary error vector which appears to be random for anyone who is not the intended receiver. The receiver, using the sparseness of her secret key, is able transform the ciphertext in such a way to significantly reduce the weight of the error vector. Then, the receiver can use the efficient decoding procedure for $\mathcal{C}$ to correct the remaining errors of the transformed ciphertext to recover the message.

The code $\mathcal{C}$ proposed by Aguilar-Melchor et al. [MAB+18] is a tensor code between a BCH code and a repetition code. One drawback of the HQC implementation submitted to NIST is that the decoder [JK95] for the BCH code is not constant-time, and depends on the weight of the error it corrects. This makes the decryption operation vulnerable to timing attacks.

The use of non-constant-time decoders has been exploited to attack code-based schemes such as QC-MDPC [ELPS18], and recently, RQC [AMBD+18], which is a variant of HQC in the rank metric that uses Gabidulin codes [Gab85], was shown vulnerable to timing attacks [BBGM19]. However, timing attacks exploiting non-constant-time decoders are not exclusive to code-base schemes, and the use of BCH codes in LAC [LLZ+18] has been shown to leak secret information from timing [DTVV19].

Contributions. We present the first timing attack on HQC. The attack follows Guo et al. [GJS16] idea: first we show how to obtain information, which is called the spectrum, on the secret key by timing a large number of decryptions, and then use the information gathered to reconstruct the key. We analyze in detail the reason behind the information leakage. As a minor contribution, we show that a randomized variant of Guo's et al. algorithm for key reconstruction is better than their recursive algorithm when the attacker has partial information on the secret key's spectrum. This is useful to reduce the number of decryption timings the attacker needs to perform.

Shortly after this paper was accepted for publication, Wafo-Tapa et al. [WTBBG19] published a preprint in the Cryptology ePrint Archive in which they also present a timing attack against HQC. Our attack is stronger in the sense that it only uses valid ciphertexts, while the attack by Wafo-Tapa et al. [WTBBG19] uses malformed ciphertexts to better control the extraction of secret information. However, their paper comes with a countermeasure, which consists of a constant-time BCH decoder with a low overhead.

Paper organization. In Section 3.2, we review some background concepts for understanding HQC and our attack. The HQC is described in Section 3.3. The attack is presented in Section 3.4. Some mathematical and algorithmic aspects of the attack are analyzed in detail in Section 3.5. In Section 3.6, we analyze the practical performance of the attack against concrete HQC parameters. We conclude in Section 3.8.

3.2 Background

Please notice that, although some of the concepts defined below already appeared in Section 2.5, the definitions are kept here to allow for readers to independently read this technical chapter.

Definition 3.2.1 (Linear codes)**.** A *binary* $[n, k]$*-linear code* is a k-dimensional linear subspace of $\mathbb{F}_2^n$, where $\mathbb{F}_2$ denotes the binary field.

Definition 3.2.2 (Generator and parity-check matrices)**.** Let $\mathcal{C}$ be a binary $[n, k]$-linear code. If $\mathcal{C}$ is the linear subspace spanned by the rows of a matrix $\mathbf{G}$ of $\mathbb{F}_2^{k\times n}$, we say that $\mathbf{G}$ is a *generator matrix* of $\mathcal{C}$. Similarly, if $\mathcal{C}$ is the kernel of a matrix $\mathbf{H}$ of $\mathbb{F}_2^{(n-k)\times n}$, we say that $\mathbf{H}$ is a *parity-check matrix* of $\mathcal{C}$.

Definition 3.2.3 (Weight)**.** The Hamming *weight* of a vector $\mathbf{v}$, denoted by $\mathrm{w}(\mathbf{v})$, is the number of its non-null entries.

Definition 3.2.4 (Support)**.** The *support* of a vector $\mathbf{v}$, denoted by $\mathrm{supp}(\mathbf{v})$, is the set of indexes of its non-null entries.

We use zero-based numbering for the vectors indexes as we believe it allows more concise descriptions in some of the algorithms and analysis.

Definition 3.2.5 (Cyclic matrix)**.** The *cyclic matrix* defined by a vector $\mathbf{v} = [v_0, \ldots, v_{n-1}]$, is the matrix

$$\mathrm{rot}(\mathbf{v}) = \begin{bmatrix} v_0 & v_{n-1} & \cdots & v_1 \\ v_1 & v_0 & \cdots & v_2 \\ \vdots & \vdots & \ddots & \vdots \\ v_{n-1} & v_{n-2} & \cdots & v_0 \end{bmatrix}.$$

Definition 3.2.6 (Vector product)**.** The *product of two vectors* $\mathbf{u}, \mathbf{v} \in \mathbb{F}_2^n$ is given as

$$\mathbf{u} \cdot \mathbf{v} = \mathbf{u}\,\mathrm{rot}(\mathbf{v})^T = \left(\mathrm{rot}(\mathbf{v})\mathbf{u}^T\right)^T = \mathbf{v}\,\mathrm{rot}(\mathbf{u})^T = \mathbf{v} \cdot \mathbf{u}.$$

Definition 3.2.7 (Syndrome decoding problem)**.** Consider the following input: a random binary matrix $\mathbf{H} \in \mathbb{F}_2^{k\times n}$, a random vector $\mathbf{s} \in \mathbb{F}_2^k$, and an integer $w > 0$. The syndrome decoding problem asks for a $\mathbf{v}$ of weight w such that $\mathbf{v}\mathbf{H}^T = \mathbf{s}$.

The quasi-cyclic syndrome decoding problem is a restriction of the syndrome decoding problem, in which $\mathbf{H}$ is a block matrix consisting of cyclic blocks.

The syndrome decoding problem is proven to be **NP**-hard [BMVT78]. Despite no complexity result on the quasi-cyclic variant, it is considered hard since all known decoding algorithms that exploit the cyclic structure have only a small advantage over general decoding algorithms for the non-cyclic case.

Definition 3.2.8 (Circular distance)**.** The *circular distance* between the indexes i and j in a vector of length n is

$$\mathrm{dist}_n(i,j) = \begin{cases} |i-j| & \text{if } |i-j| \leq \lfloor n/2 \rfloor, \\ n-|i-j| & \text{otherwise.} \end{cases}$$

We next define the spectrum of a vector, which is a crucial concept for the rest of the paper. The importance of the spectrum for the attack comes from the fact that it is precisely the spectrum of the key that can be recovered by the timing attack. Intuitively, the spectrum of a binary vector $\mathbf{v}$ is the set of circular distances that occur between two non-null entries of $\mathbf{v}$.

Definition 3.2.9 (Spectrum of a vector)**.** Let $\mathbf{v} = [v_0, v_1, \ldots, v_{n-1}]$ be an element of $\mathbb{F}_2^n$. Then the *spectrum* of $\mathbf{v}$ is the set

$$\sigma(\mathbf{v}) = \{\mathrm{dist}_n(i,j) : i \neq j,\ v_i = 1,\ \text{and } v_j = 1\}.$$

In some cases, it is important to consider the multiplicity of each distance d, that is the number of pairs of non-null entries that are at distance d apart. In such cases, we abuse notation and write $(d:m) \in \sigma(\mathbf{v})$ to denote that d appears with multiplicity m in vector $\mathbf{v}$.

Definition 3.2.10 (Mirror of a vector)**.** Let $\mathbf{v} = [v_0, v_1, \ldots, v_{n-1}]$ be an element of $\mathbb{F}_2^n$. Then the *mirror* of $\mathbf{v}$ is the vector

$$\mathrm{mirror}(\mathbf{v}) = [v_{n-1}, v_{n-2}, \ldots, v_0].$$

We sometimes abuse notation and write $\mathrm{mirror}(V)$, where V is the support of a vector $\mathbf{v}$, to represent the support of the mirror of $\mathbf{v}$. Notice that the spectrum of a vector is invariant with respect to its circular shifts and its mirror.

Guo et al. [GJS16] showed that it is possible to reconstruct a sparse vector from its spectrum. To solve this problem, they propose an algorithm that consists of a simple pruned depth-first search. Its description is given as Algorithm 3.1.[16] The main argument by Guo et al. for the efficiency of their algorithm is that unfruitful branches are pruned relatively early in the search.

Let α be the fraction of the $\lfloor n/2 \rfloor$ possible distances that are not in D, that is $\alpha = 1 - |D|/\lfloor n/2 \rfloor$. For each new level in the search tree, it is expected that a fraction α of the possible positions in the previous level survive the sieve imposed by line 15. Let MaxPaths

[16] Here we present a slightly more general version of Guo's et al. reconstruction algorithm that does not require the key's spectrum to be completely determined, but the idea is the same.

Algorithm 3.1 GJS key reconstruction algorithm [GJS16].

1: ▷ n, w the length and weight of the secret vector $\mathbf{y}$
2: ▷ D a set of distances outside $\sigma(\mathbf{y})$
3: ▷ s a distance inside $\sigma(\mathbf{y})$
4: ▷ V the partially recovered support of a shift of $\mathbf{y}$ (initially set to $\{0, s\}$, where $s \in \sigma(\mathbf{y}))$ is known
5: **procedure** GJSKeyReconstruction(n, w, D, s, V)
6: ▷ Outputs the support V of some shift of $\mathbf{y}$, or $\perp$ if $\sigma(\mathbf{y})$ is an invalid spectrum
7: **if** $|V| = w$ **then**
8: **if** V is the support of a shift of $\mathbf{y}$ **then**
9: **return** V
10: **else if** mirror(V) is the support of a shift of $\mathbf{y}$ **then**
11: **return** mirror(V)
12: **else**
13: **return** $\perp$
14: **for** each position $j = 1, \ldots, n-1$ which are not in V **do**
15: **if** $\text{dist}_n(v, j) \notin D$ for all v in V **then**
16: Add j to V
17: `ret` $\leftarrow$ GJSKeyRecovery(n, w, D, s, V) ▷ Recursive call with the updated set V
18: **if** `ret` $\neq \perp$ **then**
19: **return** V
20: Remove j from V
21: **return** $\perp$

be the total number of paths that Guo's et al. [GJS16] algorithm can explore. Then

$$\text{MaxPaths} = \prod_{\ell=2}^{w-1} \max\left(1, \lfloor n/2 \rfloor \alpha^{\ell}\right) = \lfloor n/2 \rfloor^{\phi} \alpha^{\phi(\phi+3)/2},$$

where ℓ represents the level in the search tree, and ϕ is the level at which each node has an expected number of child nodes lower than or equal to 1. Notice that, on average, the mirror test in line 10 cuts in half the number of paths the algorithm needs to explore until it finds the key. From the remaining paths, we expect that half of them have to be taken until the key is found. Therefore, considering $\mathbf{WF}_{\text{GJS}}$ to be the average number of paths the algorithm explores until a key is found, we have

$$\mathbf{WF}_{\text{GJS}} = \frac{1}{4}\text{MaxPaths} = \frac{1}{4}\lfloor n/2 \rfloor^{\phi} \alpha^{\phi(\phi+3)/2}.$$

3.3 The HQC encryption scheme

3.3.1 Setup

On input 1^{λ}, where λ is the security parameter, the setup algorithm returns the public parameters $n, k, \delta, w, w_{\mathbf{r}}, w_{\mathbf{e}}$, from parameters table such as Table 3.1. For these parameters,

an $[n, k]$ linear code $\mathcal{C}$, with an efficient decoding algorithm Ψ capable of correcting random errors of weight up to δ with overwhelming probability, is fixed. Parameters $w, w_{\mathbf{r}}$ and $w_{\mathbf{e}}$ correspond to the weights of the sparse vectors defined and used in the next sections.

Instance	Security	n_1	n_2	$n \approx n_1 n_2$ [a]	$k = k_1$	w	$w_{\mathbf{r}} = w_{\mathbf{e}}$	p_{fail}
Basic-I	128	766	29	22,229	256	67	77	2^{-64}
Basic-II	128	766	31	23,747	256	67	77	2^{-96}
Basic-III	128	796	31	24,677	256	67	77	2^{-128}
Advanced-I	192	796	51	40,597	256	101	117	2^{-64}
Advanced-II	192	766	57	43,669	256	101	117	2^{-128}
Advanced-III	192	766	61	46,747	256	101	117	2^{-192}
Paranoiac-I	256	766	77	59,011	256	133	153	2^{-64}
Paranoiac-II	256	766	83	63,587	256	133	153	2^{-128}
Paranoiac-III	256	796	85	67,699	256	133	153	2^{-192}
Paranoiac-IV	256	796	89	70,853	256	133	153	2^{-256}

Table 3.1: Suggested parameters for some security levels [AMBD+18].

[a] The value of n is the smallest prime number greater than $n_1 n_2$.

3.3.2 Key generation

Let $\mathbf{H} \in \mathbb{F}_2^{n \times 2n}$ be a quasi-cyclic matrix selected at random, in systematic form, that is $\mathbf{H} = [\,\mathbf{I}\,|\,\text{rot}(\mathbf{h})\,]$, for some vector $\mathbf{h}$. Let $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^n$ be sparse vectors with weight $\text{w}(\mathbf{x}) = \text{w}(\mathbf{y}) = w$. Compute

$$\mathbf{s} = [\mathbf{x}|\mathbf{y}]\mathbf{H}^T = \mathbf{x} + \mathbf{y} \cdot \text{rot}(\mathbf{h})^T = \mathbf{x} + \mathbf{y} \cdot \mathbf{h}.$$

The public and secret key are $\mathbf{k}_{\text{pub}} = [\mathbf{s}|\mathbf{h}]$ and $\mathbf{k}_{\text{sec}} = [\mathbf{x}|\mathbf{y}]$, correspondingly.

From this construction, it is easy to see the relation between recovering the secret key from the public key and the quasi-cyclic syndrome decoding problem.

3.3.3 Encryption

Let $\mathbf{m} \in \mathbb{F}_2^k$ be the message to be encrypted. First, choose two random sparse vectors $\mathbf{r}_1, \mathbf{r}_2 \in \mathbb{F}_2^n$ such that $\text{w}(\mathbf{r}_1) = \text{w}(\mathbf{r}_2) = w_{\mathbf{r}}$. Then choose a random sparse vector $\mathbf{e} \in \mathbb{F}_2^n$ such that $\text{w}(\mathbf{e}) = w_{\mathbf{e}}$. Let

$$\mathbf{u} = [\mathbf{r}_1|\mathbf{r}_2]\mathbf{H}^T = \mathbf{r}_1 + \mathbf{r}_2 \cdot \mathbf{h}, \text{ and } \mathbf{v} = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}.$$

Return the ciphertext $\mathbf{c} = [\mathbf{u}|\mathbf{v}]$.

3.3.4 Decryption

Compute $\mathbf{c}' = \mathbf{v} + \mathbf{u} \cdot \mathbf{y}$. Notice that

$$\begin{aligned}\mathbf{c}' &= \mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e} + (\mathbf{r}_1 + \mathbf{r}_2 \cdot \mathbf{h}) \cdot \mathbf{y} \\ &= \mathbf{mG} + (\mathbf{x} + \mathbf{y} \cdot \mathbf{h}) \cdot \mathbf{r}_2 + \mathbf{e} + (\mathbf{r}_1 + \mathbf{r}_2 \cdot \mathbf{h}) \cdot \mathbf{y} \\ &= \mathbf{mG} + \mathbf{x} \cdot \mathbf{r}_2 + \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e}.\end{aligned}$$

Intuitively, since $\mathbf{x}, \mathbf{y}, \mathbf{r}_1, \mathbf{r}_2$, and $\mathbf{e}$ all have low weight, we expect $\mathbf{e}' = \mathbf{x} \cdot \mathbf{r}_2 + \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e}$ to have a relatively low weight. This is made precise by Aguilar-Melchor et al. [AMBD+18], where they propose the public parameters to ensure that $\mathrm{w}(\mathbf{e}')$ is sufficiently low for it to be corrected out of $\mathbf{c}'$ with overwhelming probability.

Therefore we can use the decoder Ψ to correct the errors in $\mathbf{c}'$ and obtain $\mathbf{c}'' = \Psi(\mathbf{c}') = \mathbf{mG}$. We finally get $\mathbf{m}$ by solving the overdetermined linear system $\mathbf{mG} = \mathbf{c}''$.

3.3.5 Security and instantiation

In general, schemes based on syndrome decoding have to take care to avoid generic attacks based on Information Set Decoding [Pra62, Ste88, TS16]. Furthermore, the quasi-cyclic structure of the code used to secure the secret key can make the scheme vulnerable to DOOM [Sen11], or other structural attacks [GJL15, LJS+16].

To instantiate the scheme, the authors propose parameters for which they prove very low decryption error probability and resistance to the attacks mentioned. This error analysis allows the HQC to achieve IND-CCA2 security using the transformation of Hofheinz et al. [HHK17b].

Of particular interest for our timing attack, is the way that code $\mathcal{C}$ is chosen. Their proposal is to build the tensor code $\mathcal{C} = \mathcal{C}_1 \otimes \mathcal{C}_2$, where the auxiliary codes are chosen as follows. $\mathcal{C}_1$ is a BCH(n_1, k_1, δ_1) code of length n_1, dimension k_1. $\mathcal{C}_2$ is a repetition code of length n_2 and dimension 1, that can decode up to $\delta_2 = \left\lfloor \frac{n_2 - 1}{2} \right\rfloor$. Therefore, to encode a message $\mathbf{m}$ with respect to $\mathcal{C}$ is equivalent to first encode it using the BCH code $\mathcal{C}_1$, and then encode *each bit* of the resulting codeword with the repetition code $\mathcal{C}_2$.

The suggested parameters for this instantiation are shown in Table 3.1. In this table, column p_{fail} contains an upper bound for the probability of a decryption failure for each instance of the scheme. The size of the public keys and ciphertexts correspond to $2n$ bits.

3.4 Timing attack against HQC

In the decryption algorithm, the decoder Ψ is used to correct the errors in the word

$$\mathbf{c}' = \mathbf{mG} + \mathbf{x} \cdot \mathbf{r}_2 + \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e},$$

where the attacker knows every element, except for the secret key consisting of $\mathbf{x}$ and $\mathbf{y}$. For the original instantiation, where $\mathcal{C}$ is the tensor product of a BCH code and a repetition code, the decoder Ψ consists of a sequence of two operations: first apply a repetition code decoder Ψ_2, and then apply the BCH code decoder Ψ_1. That is $\Psi(\mathbf{c}') = \Psi_1(\Psi_2(\mathbf{c}'))$.

The timing attack is based on the fact the BCH decoder implemented by Aguilar-Melchor et al. [MAB+18] is not constant-time, and is slower when there are more errors to be corrected. In other words, the decryption time leaks the number of errors that the repetition code (RC) decoder Ψ_2 was not able to correct.

Figure 3.1 shows the essentially linear relation between the decryption time and the number of errors corrected by the BCH decoder. We emphasize that the time considered is for complete decryption, not only the BCH decoding step. The weight distribution is centered between 9 and 10, thus error weights larger than 22 are rare (around 1%).

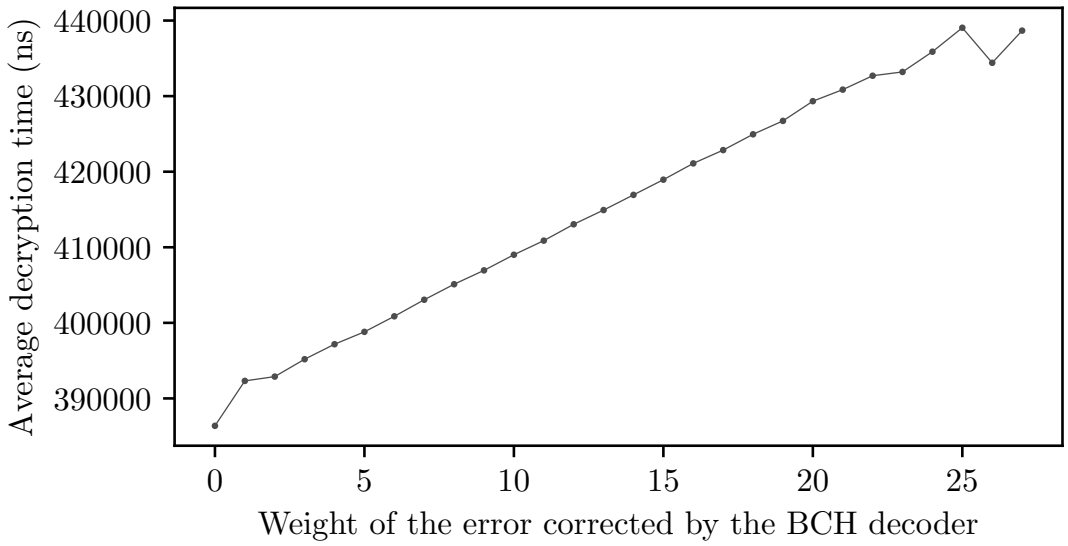

Figure 3.1: The average decryption time for different weights of the errors corrected by the BCH decoder, considering 10 million decryption operations.

Let $\mathbf{e}'$ be the error vector that Ψ_2 will try to correct, that is $\mathbf{e}' = \mathbf{x} \cdot \mathbf{r}_2 + \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e}$. We note that it is useful to consider Guo's et al. [GJS16] observation, used in their attack on QC-MDPC, that the weight of the product of two binary sparse vectors $\mathbf{a} \cdot \mathbf{b}$ is lower when the spectrums of $\mathbf{a}$ and $\mathbf{b}$ share more entries. However, this observation is not sufficient to enable us to accurately distinguish between distances in and out of the spectrum, because we are not just interested in the weight of $\mathbf{e}'$, but mainly in the probability that it has enough non-null entries in the same repetition blocks to cause Ψ_2 to leave decoding errors.

When the number of RC decoding errors for $\Psi_2(\mathbf{mG} + \mathbf{e}')$ is high, it means that $\mathbf{e}'$ has a lot of non-null entries that are at a distance lower than the repetition block size n_2. Therefore, if we understand how $\mathbf{r}_1$ and $\mathbf{y}$ influence the number of entries lower than n_2 in $\sigma(\mathbf{r}_1 \cdot \mathbf{y})$, we can use our knowledge on $\mathbf{r}_1$ together with the decryption time to obtain

information on $\mathbf{y}$.[17]

We make three observations that relate the spectrum of $\mathbf{e}'$ to the spectrums of $\mathbf{r}_1$ and $\mathbf{y}$, (alternatively $\mathbf{r}_2$ and $\mathbf{x}$). These are presented in the next section based on empirical data, and their mathematical nature is explained in Section 3.5.1.

The timing attack then consists of two parts. In the first part, called the spectrum recovery, the attacker sends Alice a great number of ciphertexts and records the decryption times for each one. This step runs until it is gathered sufficient information on the spectrums of $\mathbf{y}$ (or $\mathbf{x}$) for him to build a large set D of distances outside the spectrum, and to obtain a distance $s \in \sigma(\mathbf{y})$ (respectively, $\sigma(\mathbf{x})$). In the second part, the set D and distance s are passed to the key reconstruction algorithm.

It is important to notice that the the attacker needs only to recover one of $\mathbf{x}$ or $\mathbf{y}$, because he can use the linear relation $\mathbf{s} = \mathbf{x} + \mathbf{y} \cdot \mathbf{h}$ to easily recover one from the other.

In the next sections, the two parts are presented in detail.

3.4.1 Spectrum Recovery

This is the part where timing information is used. Let Alice be the target secret key holder. The attacker sends Alice valid ciphertexts, and records the time she takes to decrypt each challenge. Since the attacker generated all ciphertexts, then, for each one of them, he knows $\mathbf{r}_1$ and $\mathbf{r}_2$. The idea is that the attacker iteratively builds two arrays, $\mathbf{T_x}$ and $\mathbf{T_y}$, such that $\mathbf{T_x}[d]$ ($\mathbf{T_y}[d]$) is the average of the decryption time when d is in the spectrum of $\mathbf{r}_2$ (resp. $\mathbf{r}_1$).

The algorithm for spectrum recovery is given as Algorithm 3.2. Notice that we are not choosing the vectors $\mathbf{r}_1$, $\mathbf{r}_2$, which are assumed to be random. Therefore CCA2 conversions [HHK17b] do not protect the scheme against this attack.

To maximize the information obtained from each decryption timing, the proposed spectrum recovery procedure targets $\sigma(\mathbf{x})$ and $\sigma(\mathbf{y})$ simultaneously. This is interesting for the attacker since it may be the case that, after a number of challenges, the output $\mathbf{T_x}$ does not have sufficient information on $\mathbf{x}$ for it to be reconstructed, but $\mathbf{T_y}$ is sufficient to recover $\mathbf{y}$.

Figure 3.2 shows the output of the spectrum recovery algorithm $\mathbf{T_y}$ for $M = 1$ billion decryption challenges. On the left of the figure, we see that distances lower than n_2 have a significantly higher average decryption time. The figure shows that, in general, distances inside the spectrum of $\mathbf{y}$ appears to have lower average decryption time. However, there is no clear line to classify a distance d as inside or outside $\sigma(\mathbf{y})$, based only on $\mathbf{T_y}[d]$, since this value appears to also depend on the neighbors of d.

Figure 3.3 shows another interval of the same data, but with one vertical line for each distance in the spectrum. This enables us to see that regions where there are more distances inside the spectrum appear to have higher average decryption time.

[17] This sentence remains valid if we substitute $\mathbf{y}$ and $\mathbf{r}_1$ by $\mathbf{x}$ and $\mathbf{r}_2$, respectively.

Algorithm 3.2 Estimating the decryption time for each possible distance in $\sigma(\mathbf{x})$ and $\sigma(\mathbf{y})$.

▷ n, k the HQC public parameters
▷ $\mathcal{T}$ oracle that returns the target's decryption time for the challenge passed as argument
▷ M number of decoding challenges
procedure EstimateDecryptionTimeForDistances($n, k, \mathcal{T}, M$)
    $\mathbf{a}_y, \mathbf{b}_y, \mathbf{a}_x, \mathbf{b}_x \leftarrow$ zero-initialized arrays with $\lfloor n/2 \rfloor$ entries each
    for each decoding trial $i = 1, 2, \ldots, M$ **do**
        $\mathbf{m} \leftarrow$ a random message in $\mathbb{F}_2^k$
        $\mathbf{c} \leftarrow$ encryption of $\mathbf{m}$ using vectors
        $\mathbf{r}_1$ and $\mathbf{r}_2$ randomly chose
        $t = \mathcal{T}(\mathbf{c})$
        for each distance d in $\sigma(\mathbf{r}_1)$ **do**
            $\mathbf{a}_y[d] \leftarrow \mathbf{a}_y[d] + t$
            $\mathbf{b}_y[d] \leftarrow \mathbf{b}_y[d] + 1$
        for each distance d in $\sigma(\mathbf{r}_2)$ **do**
            $\mathbf{a}_x[d] \leftarrow \mathbf{a}_x[d] + t$
            $\mathbf{b}_x[d] \leftarrow \mathbf{b}_x[d] + 1$
    $\mathbf{T_x}, \mathbf{T_y} \leftarrow$ zero-initialized array with $\lfloor n/2 \rfloor$ positions
    for each distance d in $\{1, 2, \ldots, \lfloor n/2 \rfloor\}$ **do**
        $\mathbf{T_x}[d] \leftarrow \mathbf{a}_x[d]/\mathbf{b}_x[d]$
        $\mathbf{T_y}[d] \leftarrow \mathbf{a}_y[d]/\mathbf{b}_y[d]$
    ▷ $\mathbf{T_x}, \mathbf{T_y}$ are the average decryption time for distances in $\sigma(\mathbf{x})$ and $\sigma(\mathbf{y})$
    return $\mathbf{T_x}$ and $\mathbf{T_y}$

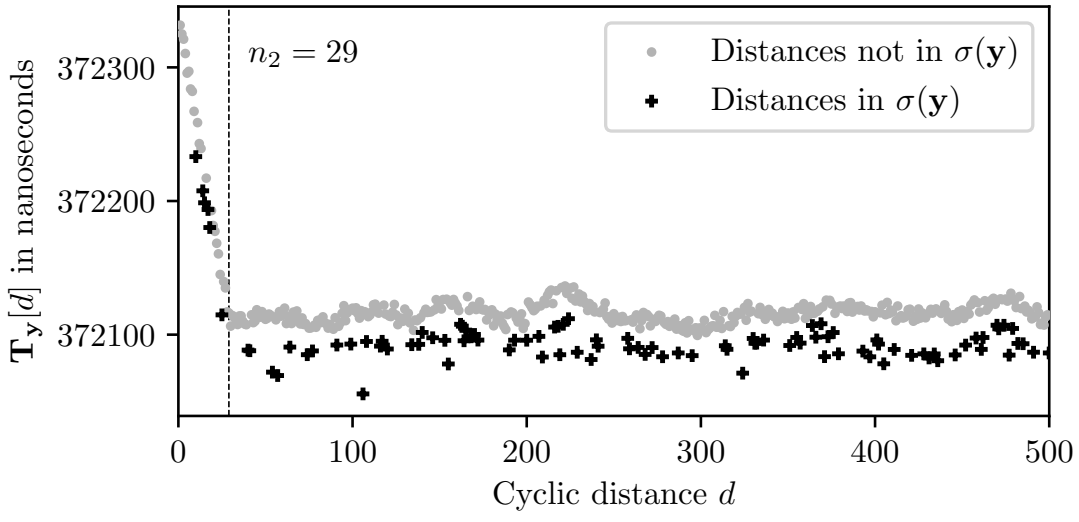

Figure 3.2: The average decryption time $\mathbf{T_y}[d]$ for each distance d that can occur in $\mathbf{r}_1$, for $M = 1$ billion.

Summarizing the analysis of the figures, we make the following three informal observations that allow us to distinguish between distances inside and outside the spectrums.

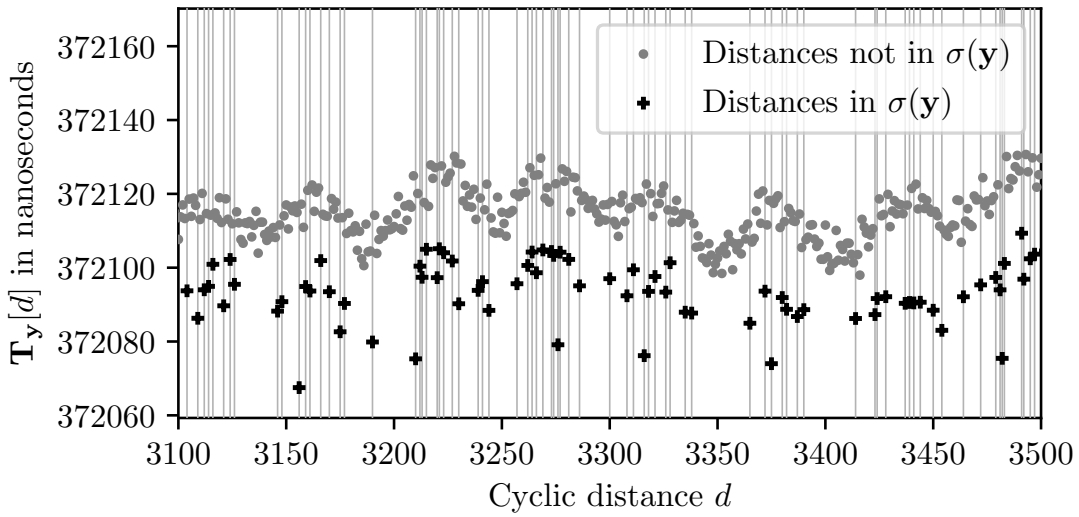

Figure 3.3: A closer look at the behavior of $\mathbf{T}_\mathbf{y}[d]$ for each distance d. The gray vertical lines represent distances inside $\sigma(\mathbf{y})$.

1. When d decreases from $d = n_2 - 1$ to $d = 1$, the value of $\mathbf{T}_\mathbf{y}[d]$ increases, getting significantly higher than the rest of the values in $\mathbf{T}_\mathbf{y}$.

2. When $d \in \sigma(\mathbf{y})$, the value of $\mathbf{T}_\mathbf{y}[d]$ is lower than the average in the neighborhood of d.

3. When d has a large number of neighbors in $\sigma(\mathbf{y})$, the value of $\mathbf{T}_\mathbf{y}[d]$ tends to be higher.

The reasons why we observe such behavior are analyzed in detail in section 3.5.1.

Similarly to the GJS algorithm (Algorithm 3.1), our key reconstruction algorithm for the next part of the attack works with two inputs: a set D of distances outside the spectrum, and a distance s inside the spectrum. Figure 3.2 suggests that, when a sufficiently large number of decryption challenges are timed, it is easy to get a distance inside the spectrum with high probability by just taking the distance s such that $\mathbf{T}_\mathbf{y}[s]$ is the minimum value in the array. However, it is not trivial to find a sufficiently large set D from $\mathbf{T}_\mathbf{y}$. For this, we propose a routine called BuildD, which is describe next.

BuildD: Building the set of distances not in $\sigma(\mathbf{y})$ from $\mathbf{T}_\mathbf{y}$.

We propose to use the following simple algorithm, that takes as input a value μ and the decryption times estimation $\mathbf{T}_\mathbf{y}$, and outputs μ distances which it classifies as out of $\sigma(\mathbf{y})$. The idea is to select the μ values of d such that $\mathbf{T}_\mathbf{y}[d]$ are among the highest of their corresponding neighborhood.

Let η be some small positive integer for which the probability that $\{d, d+1, \ldots, d+\eta-1\} \subset \sigma(\mathbf{y})$ is negligible for all values of d. The value η will be the size of the neighborhood,

which must contain at least one distance outside the spectrum. This value can be estimated by generating N random vectors, then computing the minimum value η for which η consecutive distances always contain at least one distance not in the vectors corresponding spectrums. For the Basic-I parameters, we obtained $\eta = 11$ for $N = 10000$.

For each d, we compute the difference between $\mathbf{T_y}$ [d] and the highest value of $\mathbf{T_y}$ in the window $\{d - \lfloor \eta/2 \rfloor, \ldots, d + \lceil \eta/2 \rceil - 1\}$. If the window contains invalid distances, we just truncate it to exclude them. In other words

$$\rho(d) = \left(\max_{i \in W_d} \mathbf{T_y}[i] \right) - \mathbf{T_y}[d],$$

where W_d is the intersection between $\{d - \lfloor \eta/2 \rfloor, \ldots, d + \lceil \eta/2 \rceil - 1\}$ and the set of possible distances. The algorithm sorts the possible distances with respect to $\rho(d)$, and returns the μ values of d such that $\rho(d)$ are the lowest ones.

Let $\textsc{BuildD}(\mathbf{T_y}, \mu)$ be the output of the algorithm just described for the given inputs. Since the key reconstruction only works if D is a large set of distances not in the spectrum, it is natural to define the quality of the input $\mathbf{T_y}$ as

$$\textsc{Quality}(\mathbf{T_y}) = \max \{\mu : \textsc{BuildD}(\mathbf{T_y}, \mu) \cap \sigma(\mathbf{y}) = \varnothing\}.$$

Figure 3.4 helps us visualize why this algorithm works. For $M = 1$ billion decryptions, it is easy to see that the distances between $\mathbf{T_y}[d]$ and $\max_{i \in W_d} \mathbf{T_y}[i]$ should be smaller when d is not in the spectrum of $\mathbf{y}$. However, it is not clear yet how many decryptions are necessary for the algorithm to be able to build sufficiently large sets D, that is, to obtain high values for $\textsc{Quality}(\mathbf{T_y})$. This is considered in the experimental analysis in Section 3.6.2.

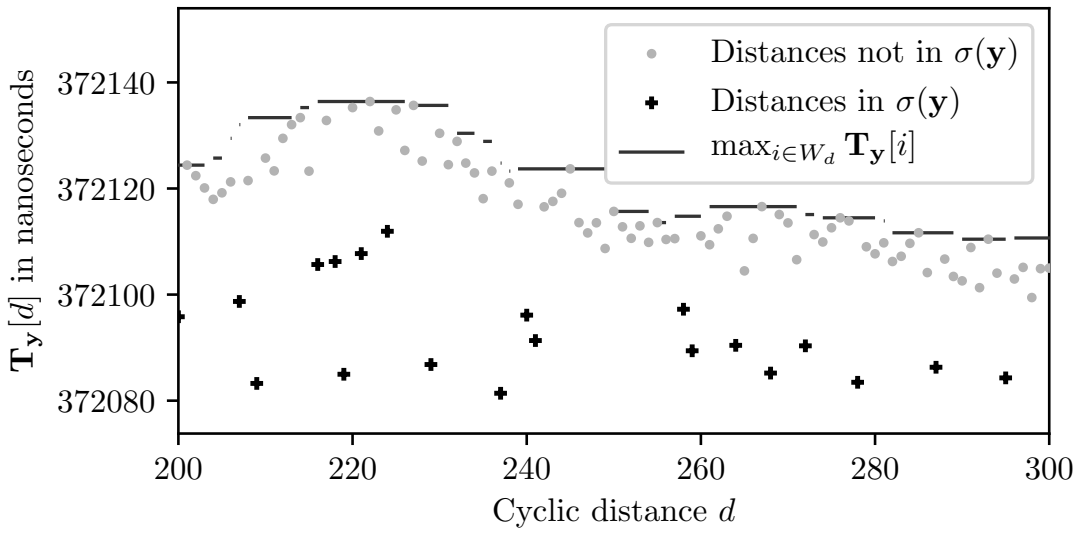

Figure 3.4: Illustration of the values $\max_{i \in W_d} \mathbf{T_y}[i]$ for each distance d, considering windows of size $\eta = 11$, after $M = 1$ billion decryptions.

3.4.2 Reconstructing y from partial information on its spectrum

We propose[18] the key reconstruction algorithm given as Algorithm 3.3, which is the simple randomized extension of Guo's et al. [GJS16] algorithm. Instead of performing a depth-first search for the key, at each level of the search tree, the algorithm chooses the next node at random.

Algorithm 3.3 Randomized key reconstruction algorithm.

1: ▷ n, w the HQC public parameters
2: ▷ s a distance inside $\sigma(\mathbf{y})$
3: ▷ D a set of distances which are not in $\sigma(\mathbf{y})$
4: **procedure** RandKeyReconstruction(n, w, s, D)
5: $V \leftarrow \varnothing$
6: **while** Both V and mirror(V) are not the support of a rotation of $\mathbf{y}$ **do**
7: $V \leftarrow \{0, s\}$
8: $\Gamma_2 \leftarrow \{i \in \{1, \ldots, \lfloor n/2 \rfloor\} - V : \mathrm{dist}_n(i, v) \notin D \text{ for all } v \in V\}$
9: $\ell \leftarrow 2$
10: **while** $|V| < w$ and $|\Gamma_\ell| > 0$ **do**
11: $p \leftarrow$ a random element from Γ_ℓ
12: $V \leftarrow V \cup \{p\}$
13: $\Gamma_{\ell+1} \leftarrow \{i \in \Gamma_\ell : \mathrm{dist}_n(i, v) \notin D \text{ for all } v \in V\}$
14: $\ell \leftarrow \ell + 1$
15: **if** V is the support of a shift of $\mathbf{y}$ **then**
16: **return** V
17: **else if** mirror(V) is the support of a shift of $\mathbf{y}$ **then**
18: **return** mirror(V)

We give a brief description of the algorithm. Parameters s, which must be a distance inside the spectrum of $\mathbf{y}$, and D, which is a set of distances outside the spectrum of $\mathbf{y}$, are obtained in the first part of the attack. At each iteration, the algorithm starts with the set $V = \{0, s\}$ and tries to complete it with $w - 2$ indexes. To complete the support V, the algorithm chooses at random an index inside the auxiliary set Γ_ℓ, which contains, for each level l, the possible positions to complete the support. That is, Γ_ℓ consists of all the elements from $\{0, \ldots, n-1\}$ which are not in V, and whose circular distance to any index in V is not in D.

Notice that it is easy to perform the tests in lines 15 and 17 without knowing the secret key. Let $\overline{\mathbf{y}}$ be the vector with support V found in the algorithm's main loop. Consider all possible cyclic shifts of $\overline{\mathbf{y}}$, denoted by $\mathbf{y}^0, \ldots, \mathbf{y}^{n-1}$. To test if $\overline{\mathbf{y}}$ is a shift of $\mathbf{y}$, we look for a shift $\mathbf{y}^i$ such that the weight of the vector $\overline{\mathbf{x}} = \mathbf{s} + \mathbf{y}^i \cdot \mathbf{h}$ is $\mathrm{w}(\overline{\mathbf{x}}) = w$. If we find one, then $\overline{\mathbf{y}}$ is a shift of $\mathbf{y}$ (high probability), or we have found an equivalent secret key for the given public key ($\mathbf{h}$, $\mathbf{s}$). If we do not find one, then we start a new iteration.

The complexity of the algorithm is analyzed in Section 3.5.2, while its practical performance is shown in Section 3.6.1.

[18] Notice that this algorithm first appeared in the author's master's thesis [Pai17, Section 6.1].

3.5 Analysis

In this section we analyze two aspects of the attack. First we explain why it is possible to distinguish between distances inside and outside the spectrum based on decryption time. Then we analyze the complexity of the randomized key reconstruction algorithm, and how it compares to the one presented by Guo et al. [GJS16].

3.5.1 Distinguishing distances inside and outside the spectrum

We know that the decryption time is related to the number of errors left by the repetition code (RC) decoder. Our main observation is that the number of RC decoding errors depends on how the spectrums of $\mathbf{r}_1$ and $\mathbf{r}_2$ relate to those of $\mathbf{y}$ and $\mathbf{x}$, respectively.

Consider the error to be corrected by the RC decoder, given by

$$\mathbf{e}' = \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{r}_2 \cdot \mathbf{x} + \mathbf{e}.$$

An RC decoding error occurs when $\mathbf{e}'$ contains more than $(n_2 - 1)/2$ nonzero errors in the same repetition block. Therefore, an RC decoding error has higher probability of occurring when the spectrum of $\mathbf{e}'$ contains small distances with high multiplicity, and in particular, when $\sigma(\mathbf{e}')$ contains a lot of distances lower than the repetition block length n_2. We also expect that $\sigma(\mathbf{e}')$ contains small distances when $\sigma(\mathbf{r}_2 \cdot \mathbf{x})$ and $\sigma(\mathbf{r}_1 \cdot \mathbf{y})$ also contain small distances. In the following discussion, we focus on $\mathbf{r}_1 \cdot \mathbf{y}$, but we could have used $\mathbf{r}_2 \cdot \mathbf{x}$ without any difference.

The above paragraph motivates us to better understand what causes the spectrum of $\mathbf{r}_1 \cdot \mathbf{y}$ to contain small distances. Unfortunately, the strong dependency between the rows of $\mathrm{rot}(\mathbf{y})^T$ can make it very hard to perform a satisfactory statistical analysis on the product $\mathbf{r}_1 \cdot \mathbf{y}$.

Therefore, we study a simpler problem, namely to describe $\sigma(\mathbf{r}_1 \cdot \mathbf{y})$ as a function of $\sigma(\mathbf{r}_1)$ and $\sigma(\mathbf{y})$, but restricted to the case where $\mathrm{w}(\mathbf{r}_1) = \mathrm{w}(\mathbf{y}) = 2$. Even though it is not the general case, it can give us a good intuition on why the attack works. The analysis is given in the following lemma. First we discuss the implications of the lemma and how it can be used to distinguish between distances inside and outside the spectrums of the secret key, and then we prove it.

Lemma 3.5.1. *Let $\mathbf{y}, \mathbf{r} \in \mathbb{F}_2^n$ be two binary vectors of weight 2, where n is an odd prime. Let α and β be the only distances in $\sigma(\mathbf{y})$ and $\sigma(\mathbf{r})$, respectively. Then, we have the following possibilities.*[19]

If $\alpha = \beta$, then

$$\sigma(\mathbf{r} \cdot \mathbf{y}) = \{\mathrm{dist}_n(0, 2\alpha) : 1\} = \{\mathrm{dist}_n(0, 2\beta) : 1\}. \tag{3.1}$$

[19] Recall that we use $(\gamma : m) \in \sigma(\mathbf{y})$ to denote that cyclic distance γ occurs m times between non-null entries of $\mathbf{y}$.

If $\alpha \neq \beta$, then

$$\sigma(\mathbf{r} \cdot \mathbf{y}) = \{\alpha : 2, \tag{3.2}$$
$$\beta : 2, \tag{3.3}$$
$$|\beta - \alpha| : 1, \tag{3.4}$$
$$\text{dist}_n(0, \beta + \alpha) : 1\}. \tag{3.5}$$

Interpreting Lemma 3.5.1. Intuitively, α represents distances inside the spectrum of the secret vector $\mathbf{y}$, while β represents distances inside the spectrum of $\mathbf{r}_1$. We now restate the observations from Section 3.4.1 with brief discussions on why they happen, using the lemma to help us.

1. When β decreases from $\beta = n_2 - 1$ to $\beta = 1$, the value of $\mathbf{T}_\mathbf{y}[\beta]$ increases, getting significantly higher than the rest of the values in $\mathbf{T}_\mathbf{y}$.

   From (3.3), distance β in $\sigma(\mathbf{r}_1)$ can cause $\sigma(\mathbf{r}_1 \cdot \mathbf{x})$ to contain β with multiplicity 2. Therefore when $\beta < n_2$, it can be responsible for more RC errors than values of $\beta \geq n_2$. The reason why $\mathbf{T}_\mathbf{y}[\beta]$ gets increasingly higher when β approaches 1 is that, we get an increasing incidence of $\beta + \alpha < n_2$, where $\alpha \in \sigma(\mathbf{y})$. Therefore, from (3.5), these values of β tend to cause more distances lower than n_2 in $\sigma(\mathbf{r}_1 \cdot \mathbf{y})$.

2. When $\beta \in \sigma(\mathbf{y})$, the value of $\mathbf{T}_\mathbf{y}[\beta]$ is lower than the average in the neighborhood of β.

   Comparing both cases considered by the lemma, we see that values of $\beta = \alpha$ for some $\alpha \in \sigma(\mathbf{y})$ (Case 1) are expected to produce a lower number of small distances in $\sigma(\mathbf{r}_1 \cdot \mathbf{x})$ than values of $\beta \neq \alpha$ for all $\alpha \in \sigma(\mathbf{y})$ (Case 2).

3. When β has a large number of neighbors in $\sigma(\mathbf{y})$, the value of $\mathbf{T}_\mathbf{y}[\beta]$ tends to be higher.

   Using (3.4), we have that $\mathbf{T}_\mathbf{y}[\beta]$ tends to be higher when more values of $\alpha \in \sigma(\mathbf{y})$ satisfy $|\beta - \alpha| < n_2$. In fact, the lemma even helps us formalize the neighborhood of β as the distances d between $\beta - n_2 < d < \beta + n_2$.

We now proceed with the proof of Lemma 3.5.1.

Lemma 3.5.1. Let α_1, α_2 and β_1, β_2 be the positions of the two ones in $\mathbf{y}$ and $\mathbf{r}$, respectively. We can suppose without loss of generality that

$$\alpha_2 = \alpha_1 + \alpha \bmod n, \text{ and } \beta_2 = \beta_1 + \beta \bmod n,$$

since if this is not the case, we can just swap the corresponding values.

The product $\mathbf{r} \cdot \mathbf{y}$ consists of the sum of two circular shifts of $\mathbf{y}$: one by β_1, and the other of β_2 positions, denoted by $\text{shift}_{\beta_1}(\mathbf{y})$ and $\text{shift}_{\beta_2}(\mathbf{y})$, respectively. More formally

$$\mathbf{r} \cdot \mathbf{y} = \mathbf{r}\,\text{rot}(\mathbf{y})^T = \text{shift}_{\beta_1}(\mathbf{y}) + \text{shift}_{\beta_2}(\mathbf{y}),$$

where

$$\begin{aligned}\text{supp}\left(\text{shift}_{\beta_1}(\mathbf{y})\right) &= \{\alpha_1 + \beta_1 \bmod n, \alpha_2 + \beta_1 \bmod n\} \\ &= \{\alpha_1 + \beta_1 \bmod n, \alpha_1 + \alpha + \beta_1 \bmod n\},\end{aligned}$$

and

$$\begin{aligned}\text{supp}\left(\text{shift}_{\beta_2}(\mathbf{y})\right) &= \{\alpha_1 + \beta_2 \bmod n, \alpha_2 + \beta_2 \bmod n\} \\ &= \{\alpha_1 + \beta_1 + \beta \bmod n, \alpha_1 + \alpha + \beta_1 + \beta \bmod n\}.\end{aligned}$$

Therefore the weight of $\mathbf{r} \cdot \mathbf{y}$ is at most 4, but can be lower if the supports above share some of their entries. We consider separately the cases when $\alpha = \beta$ and $\alpha \neq \beta$. These cases are illustrated in Figure 3.5.

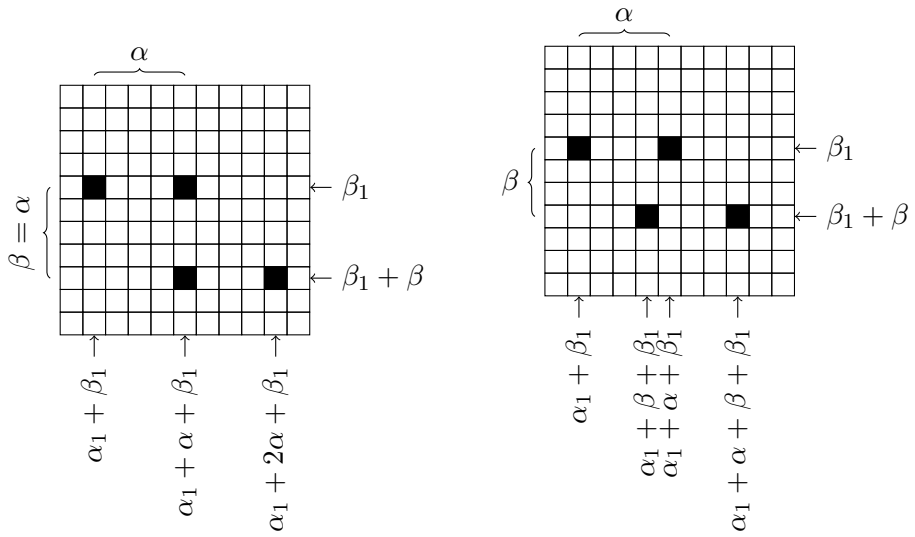

Figure 3.5: The cases when $\alpha = \beta$ (left), and $\alpha \neq \beta$ (right).

Case $\alpha = \beta$. In this case, we have:

$$\text{supp}\left(\text{shift}_{\beta_1}(\mathbf{y})\right) = \{\alpha_1 + \beta_1 \bmod n, \alpha_1 + \alpha + \beta_1 \bmod n\},$$

and

$$\text{supp}\left(\text{shift}_{\beta_2}(\mathbf{y})\right) = \{\alpha_1 + \beta_1 + \alpha \bmod n, \alpha_1 + 2\alpha + \beta_1 \bmod n\}.$$

The supports of the shifts share the entry $\alpha_1 + \beta_1 + \alpha \bmod n$. But notice that this is the only shared entry, since the fact that n is odd implies $\alpha_1 + \beta_1 \not\equiv \alpha_1 + \beta_1 + 2\alpha \mod n$. Then, summing the shifts of $\mathbf{y}$ we get

$$\operatorname{supp}(\mathbf{r} \cdot \mathbf{y}) = \{\alpha_1 + \beta_1 \bmod n, \alpha_1 + 2\alpha + \beta_1 \bmod n\}.$$

Therefore, using the facts that $\alpha \leq \lfloor n/2 \rfloor$ and n is odd, we get

$$\begin{aligned}\sigma(\mathbf{r} \cdot \mathbf{y}) &= \{\operatorname{dist}_n(\alpha_1 + \beta_1 \bmod n, \alpha_1 + 2\alpha + \beta_1 \bmod n) : 1\} \\ &= \{\operatorname{dist}_n(0, 2\alpha \bmod n) : 1\} \\ &= \{\operatorname{dist}_n(0, 2\alpha) : 1\}.\end{aligned}$$

Case $\alpha \neq \beta$. We begin by showing that the supports of the shifts do not share any entry. It is clear that $\alpha_1 + \beta_1$ is not equivalent to $\alpha_1 + \beta_1 + \beta$ nor $\alpha_1 + \alpha + \beta_1 + \beta \pmod n$ since $1 < \alpha, \beta \leq (n-1)/2$. The same can easily be seen for $\alpha_1 + \alpha + \beta_1$.

Therefore, spectrum of $\mathbf{r} \cdot \mathbf{y}$ consists of the following distances.

1. $\operatorname{dist}_n(\alpha_1 + \beta_1, \alpha_1 + \beta_1 + \beta) = \operatorname{dist}_n(0, \beta) = \beta$.
2. $\operatorname{dist}_n(\alpha_1 + \beta_1, \alpha_1 + \alpha + \beta_1) = \operatorname{dist}_n(0, \alpha) = \alpha$.
3. $\operatorname{dist}_n(\alpha_1 + \beta_1, \alpha_1 + \alpha + \beta_1 + \beta) = \operatorname{dist}_n(0, \alpha + \beta)$.
4. $\operatorname{dist}_n(\alpha_1 + \beta_1 + \beta, \alpha_1 + \alpha + \beta_1) = \operatorname{dist}_n(\beta, \alpha) = \operatorname{dist}_n(0, \alpha - \beta) = |\alpha - \beta|$.
5. $\operatorname{dist}_n(\alpha_1 + \beta_1 + \beta, \alpha_1 + \alpha + \beta_1 + \beta) = \operatorname{dist}_n(0, \alpha) = \alpha$.
6. $\operatorname{dist}_n(\alpha_1 + \alpha + \beta_1, \alpha_1 + \alpha + \beta_1 + \beta) = \operatorname{dist}_n(0, \beta) = \beta$.

Counting the multiplicities of these distances, we get the desired result. □

3.5.2 Probabilistic analysis of the key reconstruction algorithm

In this section[20], we first analyze our randomized variant of the key reconstruction algorithm, given as Algorithm 3.3 in Section 3.4.1. We then compare it to Guo's et al. recursive algorithm, described as Algorithm 3.1 in the end of Section 3.2.

In each iteration, the algorithm performs a random walk down the search tree, starting from the root $\{0, s\}$, corresponding to $\ell = 2$, and ending in one of its leaves. Therefore, for the algorithm to succeed in finding $\mathbf{y}$, it has to choose, in each level of the search, an element in $\operatorname{supp}(\mathbf{y})$.

Let s be a distance in $\sigma(\mathbf{y})$. Suppose the search is at level ℓ, and the algorithm has chosen, until now, the elements $V_\ell = \{v_1 = 0, v_2 = s, \ldots v_\ell\}$, all in the support of $\mathbf{y}$. Let Γ_ℓ be the set of possible choices at level ℓ, then

$$\Gamma_\ell = \{p \in (\{0, \ldots, n-1\} - V_\ell) : \operatorname{dist}_n(p, v) \notin D \text{ for all } v \in V\}.$$

[20] This analysis is adapted from the author's master's thesis [Pai17, Section 6.2].

We now have exactly $w - |V_\ell|$ good choices for the next level, which gives us

$$\Pr\left(v_{\ell+1} \in \operatorname{supp}(\mathbf{y}) \mid V_\ell \subset \operatorname{supp}(\mathbf{y})\right) = \frac{w - |V_\ell|}{|\Gamma_\ell|} = \frac{w - \ell}{|\Gamma_\ell|}.$$

Remember that the spectrum recovery algorithm can find either $\mathbf{y}$ or mirror($\mathbf{y}$), and both are of interest to the attacker. Therefore, we can write the probability that the algorithm successfully finds the key as

$$\Pr(\text{Success}) = 2 \prod_{\ell=2}^{w-1} \frac{w - \ell}{|\Gamma_\ell|},$$

where the product starts at level $\ell = 2$ since the search begins with $V_2 = \{0, s\}$, and it ends at level $\ell = w - 1$ because this is the last level in which a choice is made. The factor 2 comes from the mirror test.

Unfortunately, it is not easy to compute the distribution of $|\Gamma_\ell|$, because of the dependency between distances in D and elements in V_ℓ. However, we can approximate its expected value using an argument similar to the one used by Guo et al. [GJS16]. Let α be the probability that a distance is not in D, that is $\alpha = 1 - |D|/\lfloor n/2 \rfloor$. At level ℓ, there are $w - \ell$ choices that are in $\operatorname{supp}(\mathbf{y})$, and ℓ positions already in V_ℓ. For the other $n - w$ positions that are not in the support of $\mathbf{y}$, we expect a fraction of α^ℓ of them to have survived the sieves of each level. Therefore, we have

$$\mathbb{E}\left(|\Gamma_\ell|\right) \approx (n - w)\alpha^\ell + w - \ell.$$

We define the work factor $\mathbf{WF}_{\text{RAND}}$ of this algorithm as the expected number of paths it needs to explore until it finds the secret key. Then, using the approximation above, its value is

$$\begin{aligned}\mathbf{WF}_{\text{RAND}} &= \frac{1}{\Pr(\text{Success})} \\ &\approx \frac{1}{2} \prod_{\ell=2}^{w-1} \frac{(n - w)\alpha^\ell + w - \ell}{w - \ell} = \frac{1}{2} \prod_{\ell=2}^{w-1} \left(\frac{(n - w)\alpha^\ell}{w - \ell} + 1 \right).\end{aligned}$$

Looking at the term in each level ℓ, they appear to be lower than the corresponding ones for Guo's et al. algorithm. However, just looking at the expressions, it is not clear how they compare.

To better understand how they compare, consider Figure 3.6, which shows a concrete comparison of the work factors for both algorithms when the input D has an increasing number of distances outside the spectrum. We considered parameters for three HQC variants. Since the range of $|D|$ varies according to the parameters n and w, we normalized its value with respect to the average of the total number of distances outside the spectrum, denoted by Δ. To estimate Δ for each pair (n, w), we generated 1000 different random vectors and computed the average number of distances outside the spectrums. We can

see that the work factor of the randomized algorithm is typically more than 3 orders of magnitude lower than Guo's et al. [GJS16] recursive one.

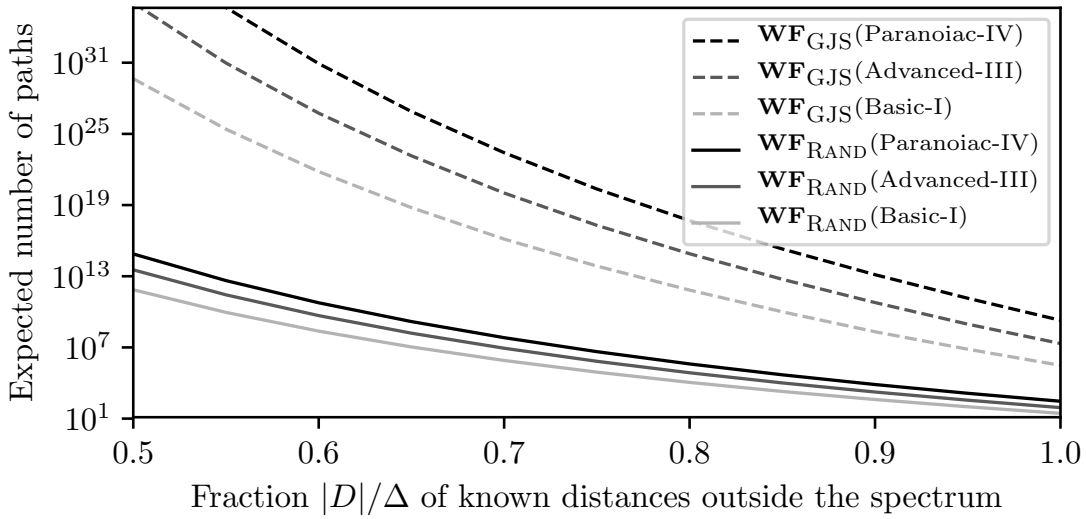

Figure 3.6: Comparison between Guo's et al. [GJS16] key reconstruction algorithm and our randomized variant with respect to the expected number of paths until the secret key is found. For each set of parameters (n, w) the value of $|D|$ is normalized using the average number of distances outside the spectrum, denoted by Δ.

3.6 Experimental results

In this section, we present our results for the timing attack against the Basic-I parameters of the HQC. We consider the two parts of the attack separately. First we run experiments on the key reconstruction algorithm to find out how much information on the spectrum it needs to run efficiently. We then run simulations to estimate how many decryptions timings an attacker needs to perform to be able to reconstruct the key. The source code and data are available at www.ime.usp.br/~tpaiva.

3.6.1 Performance of the Key Reconstruction Algorithm

We want to determine how many entries outside the spectrum of the secret vector $\mathbf{y}$ the attacker needs to know for the key reconstruction algorithm to efficiently reconstruct the vector. In other words, we are interested in how large the set D needs to be. Figure 3.6 gives us a hint on this matter, but it does not give us a concrete estimation of the key reconstruction algorithm's performance.

Table 3.2 shows the performance of both key reconstruction algorithms, the GJS and our randomized variant, when given sets D of different sizes for the Basic-I HQC parameters. For each considered size for the set D, we generated 10 random secret keys and considered D as a random set of $|D|$ distances outside the spectrum. The distance s was selected at

random from the secret key spectrum. For a more clear interpretation of the results, we considered, in the second column, the approximate average number $\Delta = 9104$ of distances not in the spectrum, to normalize the values $|D|$. We then ran C implementations of the algorithms with parameters D and s. This experiment was performed on an Intel i7-8700 CPU at 3.20GHz, using its 12 hyperthreads.

		Randomized variant of the GJS reconstruction algorithm			**GJS reconstruction algorithm**
$\|D\|$	$\|D\|/\Delta$	$\mathbf{WF}_{\text{RAND}}$	Median of the number of paths	Median of the CPU time (s)	Median of the CPU time (s)
9104	100%	28	63	0.51	0.98
8648	95%	99	80	0.51	10.78
8192	90%	407	232	0.50	772.64
7736	85%	1957	1714	0.75	6801.10
7280	80%	11394	9995	1.96	-
6824	75%	83670	54721	10.02	-
6368	70%	816671	365604	75.63	-
5912	65%	11355108	8472060	2767.90	-
5456	60%	246873607	-	-	-

Table 3.2: Performance of the key reconstruction algorithms when input D has different sizes, for the Basic-I HQC parameters.

We can see that our randomized algorithm performs much better than Guo's et al. [GJS16] one. This not only implies that the randomized algorithm allows faster key reconstruction, but also that it allows the attacker to recover the key with less interaction with the secret key holder. The estimates for the number of paths $\mathbf{WF}_{\text{RAND}}$ appear to be sufficiently accurate for our purposes, with only a minor discrepancy when $D/\Delta = 100\%$ that happens because of the concurrent hyperthreads. From $D/\Delta = 60\%$ down, the randomized algorithm starts taking too long to finish. Therefore, we consider that we are able to efficiently reconstruct the key when $D/\Delta \geq 65\%$.

3.6.2 Communication Cost

We now analyze how many decryption challenges an attacker needs to send to the secret key holder for a successful attack. In this paper, we only considered the Basic-I HQC parameters, but this experiment can easily be extended for the other parameters.

For the analysis, 10 secret keys were generated at random, and for each of them we ran the spectrum recovery algorithm for $M = 700$ million challenges. For each number challenges i, consider the quality of the decryption time estimates $\mathbf{T}^i_{\mathbf{x}}$ and $\mathbf{T}^i_{\mathbf{y}}$, given by

$$\begin{aligned}
\text{QUALITY}(\mathbf{T}^i_{\mathbf{x}}) &= \max\left\{\mu : \text{BUILDD}(\mathbf{T}^i_{\mathbf{x}}, \mu) \cap \sigma(\mathbf{x}) = \varnothing\right\}, \text{ and} \\
\text{QUALITY}(\mathbf{T}^i_{\mathbf{y}}) &= \max\left\{\mu : \text{BUILDD}(\mathbf{T}^i_{\mathbf{y}}, \mu) \cap \sigma(\mathbf{y}) = \varnothing\right\},
\end{aligned}$$

where BUILDD is the algorithm described in the end of Section 3.4.1. For the BUILDD

procedure, we considered the window size $\eta = 11$, which was obtained by the independent simulation described in the end of Section 3.4.1.

Based on the results from the previous section, we consider that the key reconstruction algorithm can efficiently recover $\mathbf{x}$ or $\mathbf{y}$, when either $\textsc{Quality}(\mathbf{T}^i_{\mathbf{x}})$ or $\textsc{Quality}(\mathbf{T}^i_{\mathbf{y}})$ is greater than 5912, correspondingly.

Figure 3.7 shows the result of the experiment. We can see that with about 400 million of challenges, efficient key reconstruction is possible. After 600 million challenges, almost all distances outside the spectrum can be correctly identified.

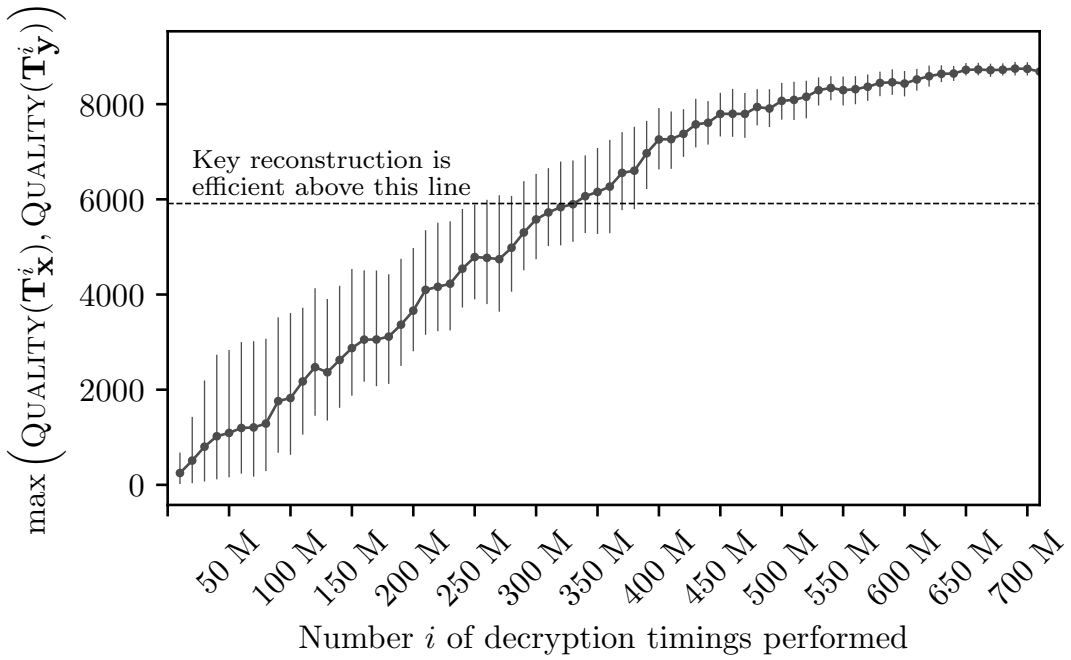

Figure 3.7: Number of decryption timings an attacker needs to perform before the key can be successfully reconstructed. A confidence level of 95% was considered for the error bars.

3.7 Discussion on countermeasures

The most obvious countermeasure against this timing attack is to use constant-time BCH decoders [WR19, WTBBG19]. However, these decoders were proposed recently and they are not well studied yet. As such, their security against other types of side-channel attacks needs further investigation.

Walters and Sinha Roy [WR19] studied constant-time BCH decoders in the context of LAC [LLZ+18]. Their decoder yields overheads between 10% and 40% when used in LAC. The optimized decoder proposed by Wafo-Tapa et al. [WTBBG19] can yield reasonable overheads, between 3% and 11%, for the different security levels provided by the HQC instances. Hopefully, with further study on constant-time BCH decoders, lower overheads can be achieved.

If the slowdown factor is a problem, one could try to add a number of errors to the partially decoded vector right before the BCH decoding procedure. Next, we explain the rationale behind this idea. Consider the vector $\mathbf{c}' = \mathbf{mG} + \mathbf{x} \cdot \mathbf{r}_2 + \mathbf{r}_1 \cdot \mathbf{y} + \mathbf{e}$. When applying the repetition code decoder to each block of n_2 elements of $\mathbf{c}'$, we can estimate the probability of a repetition decoding error from the number of ones (or zeros) in the block. For example, if the number of 1's and 0's in a block are similar (both close to $n_2/2$), then the probability of a decoding error to occur is high. This might make it possible to estimate, within some statistical margin, the number of errors that the repetition code decoder has left for the BCH decoder. Then, one can add intentional errors to the partially decoded vector $\mathbf{c}'' = \Psi_2(\mathbf{c}')$, for it to have a weight W, where W is a constant error weight which the BCH decoder can correct. Further study and a careful probabilistic analysis is needed to understand if a decoder using this idea is secure.

3.8 Conclusion

In this paper, we present the first timing attack on the HQC encryption scheme. The attack depends on the choice of the parameter code $\mathcal{C}$ and its decoder implementation. We show that the attack is practical, requiring about 400 million decryption timings to be performed. This makes the use of constant-time decoders for $\mathcal{C}$ mandatory.

We discuss possible countermeasures against this timing attack, with the preferred one being to use constant-time BCH decoders [WR19]. However, further study is needed for the secure and efficient adoption of these decoders. Other solution would be to use codes for which efficient constant-time decoders are known. One interesting future work would be to find alternatives for the code $\mathcal{C}$ that admit compact keys and efficient constant-time decoders, and for which we can prove negligible decryption failure probability.

Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. This research is part of the INCT of the Future Internet for Smart Cities funded by CNPq proc. 465446/2014-0, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, FAPESP proc. 14/50937-1, and FAPESP proc. 15/24485-9.

Chapter 4

Cryptanalysis of the binary permuted kernel problem

Abstract. In 1989, Shamir presented an efficient identification scheme (IDS) based on the permuted kernel problem (PKP). After 21 years, PKP was generalized by Lampe and Patarin, who were able to build an IDS similar to Shamir's one, but using the binary field. This binary variant presented some interesting advantages over Shamir's original IDS, such as reduced number of operations and inherently resistance against side-channel attacks. In the security analysis, considering the best attacks against the original PKP, the authors concluded that none of these existing attacks appeared to have a significant advantage when attacking the binary variant. In this paper, we propose the first attack that targets the binary PKP. The attack is analyzed in detail, and its practical performance is compared with our theoretical models. For the proposed parameters originally targeting 79 and 98 bits of security, our attack can recover about 100% of all keys using less than 2^{63} and 2^{77} operations, respectively.
Keywords: permuted kernel problem, cryptanalysis, post-quantum cryptography

4.1 Introduction

With the engineering progress on building larger quantum computers, the main cryptographic schemes used today become more and more vulnerable. Since 2016, the National Institute of Standards and Technology (NIST), is running a standardization process for post-quantum cryptography [CCJ+16]. A similar initiative is conducted by the Chinese Association for Cryptographic Research (CACR).

One of the candidate for CACR's competition is PKP-DSS [BFK+19], a digital signature scheme based on the hardness of the permuted kernel problem (PKP). This signature scheme is obtained by applying the Fiat-Shamir [FS86] transform on Shamir's PKP-based identification scheme[Sha89], which dates back from 1989. Given a matrix $\mathbf{A}$ and a vector $\mathbf{v}$ with elements in a finite field, PKP asks to find a permutation of the entries of $\mathbf{v}$ that is in the kernel of $\mathbf{A}$. PKP is NP-hard and there is no known quantum algorithm which

have a significant advantage over classical algorithms when solving the problem.

In 2010, Lampe and Patarin proposed a generalized version of PKP, in which vector $\mathbf{v}$ is substituted by a matrix $\mathbf{V}$. This enabled them to instantiate PKP in the binary field, without an apparent security loss. At the time, this binary variant presented some interesting advantages such as a reduction in the number of operations and an inherently resistance to side-channel attacks. To estimate the security of binary PKP, the authors considered the best attacks against the original PKP, with minor adjustments to make they work against the binary variant. They noted that none of the available attacks was significantly faster against binary PKP.

However, the use of binary coefficients for matrix $\mathbf{A}$ comes with a security risk. We observed that that low weight binary words occur with non-negligible probability in two public spaces: one is generated by the matrix $\mathbf{A}$ while the other is generated by the kernel of $\mathbf{V}$. It is then possible to devise an attack against binary PKP by matching these low weight codewords using subgraph isomorphism algorithms, and recovering the secret permutation from these matchings.

Contribution. In this paper, we present the first attack that specifically targets the binary PKP. Unlike previous attacks, which need a very large amount of memory to run efficiently, our attack uses only a negligible amount of memory. This allows us to provide a concrete implementation of the attack. We provide a detailed analysis of the attack, and then compare these results with the attack's performance in practice. As an example of the power of the attack: for binary PKP parameters originally targeting 80 bits of security, it uses about 2^{63} CPU cycles to fully recover the key, while the best previously known attack [KMRP19] needs about 2^{76} matrix-vector multiplications and 2^{50} bytes of memory.

Paper organization. In Section 4.2, we introduce our notation and review basic concepts of Coding Theory. Then, PKP and its binary variant are presented in Section 4.3, where we also review previous attacks against PKP. The attack is described in Section 4.4 and its performance is analyzed in Section 4.5. The asymptotic analysis of the attack is given in Section 4.6. In Section 4.7 we briefly describe how to choose secure parameters for binary PKP. In Section 4.8 we conclude and provide directions for future work.

4.2 Background

This section introduces the notation and reviews important concepts in Coding Theory. Please notice that, although some of the concepts defined below already appeared in Section 2.5, the definitions are kept here to allow for readers to independently read this technical chapter.

Notation. Vectors and matrices are denoted by lower and upper case bold letters, respectively. In general, vectors are rows, except when explicitly mentioning specific columns

of matrices. If ϕ is a permutation of n elements and $\mathbf{M}$ is an $n \times n$ matrix, then $\mathbf{M}_\phi$ and $\phi(\mathbf{M})$ correspond to the action of permutation ϕ over rows and columns of $\mathbf{M}$, respectively. For any matrix $\mathbf{X}$, we denote its i–th column as $(\mathbf{X})^i$. We denote the finite field of q elements as $\mathbb{F}_q$.

We abuse the factorial notation to avoid overloading expressions in the analysis of the attack. For any $x \in [0, +\infty)$, we let

$$x! = \begin{cases} \Gamma(x+1), \text{ if } x \geq 1, \text{ and} \\ 1, \text{ otherwise.} \end{cases}$$

Clearly it does not affect the definition of factorials of integers. Furthermore, it allows us to evaluate upper bounds of products of factorials of real numbers without having to worry about the interval $x \in (0, 1)$, where $\Gamma(x+1) < 1$, which could make the product vanish rapidly. Using this notation, we can then write $\binom{x}{y} = x!/((x-y)!y!)$, for $x, y \in [0, +\infty)$ with $x > y$. These will make for a more clear description of our approximations in Section 4.5.

Coding Theory. A binary $[n, k]$-linear code is a k-dimensional linear subspace of $\mathbb{F}_2^n$, where $\mathbb{F}_2$ denotes the binary field. Let $\mathcal{C}$ be a binary $[n, k]$-linear code. If $\mathcal{C}$ is the linear subspace spanned by the rows of a matrix $\mathbf{G}$ in $\mathbb{F}_2^{k \times n}$, we say that $\mathbf{G}$ is a generator matrix of $\mathcal{C}$. The Hamming weight of a vector $\mathbf{v}$, denoted by $\mathrm{w}(\mathbf{v})$, is the number of its non-null entries. The support of a vector $\mathbf{v}$, denoted by $\mathrm{supp}(\mathbf{v})$, is the set of indexes of its non-null entries.

4.3 The permuted kernel problem

Let us begin by formally defining the permuted kernel problem. Let $\mathbf{A}$ be an $m \times n$ matrix and $\mathbf{v}$ be a vector of n entries whose coordinates are taken from a finite field $\mathbb{F}_p$. Then, the permuted kernel problem asks to find some permutation π of the coordinates of $\mathbf{v}$ such that $\mathbf{A}\mathbf{v}_\pi^\top = \mathbf{0}$.

PKP is well-known to be NP-hard [GJ79], and it is conjectured to be hard on the average case. The naive approach to solve this problem would be to test all permutations of the entries of $\mathbf{v}$. Intuitively, there are two components which make the problem hard. The first is the large number of possible permutations, which is close to $n!$, when $\mathbf{v}$ does not have a large number of equal entries. The second is the small number of permutations of $\mathbf{v}$ which are in the kernel of $\mathbf{A}$.

In 2011, Lampe and Patarin [LP12] considered a PKP variant with $p = 2$. The authors pointed a few problems when transitioning to the binary setting that need to be taken into account. One is that the number of different permutations is significantly reduced, since every two binary vectors of the same weight are equal, up to some permutation. Furthermore, for a fixed matrix $\mathbf{A}$, there are effectively only n possibilities for $\mathbf{v}$, corresponding to one for each possible value of $\mathrm{w}(\mathbf{v})$. To avoid these problems, they proposed the use of

an $n \times \ell$ matrix $\mathbf{V}$ instead of the vector $\mathbf{v}$, obtaining the following PKP variant.

Definition 4.3.1 (Binary PKP [LP12])**.** Let $\mathbf{A}$ be an $m \times n$ binary matrix and $\mathbf{V}$ be an $n \times \ell$ binary matrix. Then, the permuted kernel problem asks to find some permutation π of the rows of $\mathbf{V}$ such that $\mathbf{AV}_\pi = \mathbf{0}$.

Notice that the original PKP can be seen as an instance of this generalized variant, by taking p instead of 2, and $\ell = 1$.

Even though the main interest on PKP is for the construction of signature schemes, we will not review details of Shamir's protocol [Sha89] or PKP-DSS [BFK+19] this construction because they are not relevant for our attack.

4.3.1 Previous attacks on PKP

After Shamir introduced the PKP-based IDS [Sha89], there has been some effort to find efficient algorithms to solve the problem. In 1990, Georgiades [Geo92] discussed how one can use symmetric equations, such as the sum of the entries of $\mathbf{v}$ or the sum of their squares, can help in lowering the number of permutations one needs to test. This, combined with the linear relations among the coordinates of kernel elements, can reduce the number of permutations to test in a brute force attack from $n!$ to $n!/(m+2)!$ permutations.

Soon after, in 1992, Baritaud et al. [BCCG92] proposed a time-memory tradeoff, where one first precompute a large table of partial solutions, which is then used to speed up a bruteforce search. In particular, for attack parameters (k, k'), their algorithm searches for solutions of a set of $k \leq m$ equations, after precomputing partial values of the equations when some set of k' variables are fixed by some arrangement of the entries in $\mathbf{v}$.

In 1993, Patarin and Chauvaud [PC93] showed a significant improvement on the cryptanalysis of PKP, which is also based on a time-memory tradeoff. Their idea was to partition the variables of the linear equation $\mathbf{Av}_\pi^\top = \mathbf{0}$ into two sets. For one set, all the possible values for their linear combination is computed and stored in a file. Then, a brute-force search, which is sped-up by the precomputed values, is used to find the values of the other set of variables. Furthermore, in 1997, Poupard [Pou97] provided a careful and realistic extension on the analysis of Patarin and Chauvaud's algorithm by considering the impact of reasonable memory limitations on the time-memory trade-off.

In 2001, Jaulmes and Joux [JJ01] proposed a new attack against PKP, which is also based on a time-memory tradeoff, but used a very different strategy. Their attack consists in adapting an algorithm for counting points in an elliptic curve [JL01] to solve a new problem, called 4SET, to which PKP can be reduced. Interestingly, this approach resulted in an algorithm somewhat similar to the one by Patarin and Chavaud [PC93], but, for years after the attack was published, it appeared to be more efficient.

More recently, in 2019, Koussa, Macario-Rat and Patarin [KMRP19] presented two important contributions on the hardness of PKP. Their first contribution is to provide a detailed analysis of the attack proposed by Jaulmes and Joux [JJ01], which was considered to be the most efficient attack against PKP. They concluded that Jaulmes and Joux's attack

Parameter set	Security level	Targeted security level when proposed	p	n	m	ℓ
BPKP–76 [LP12]	76	79	2	38	15	10
BPKP–89 [LP12]	89	98	2	42	15	11
PKP–128 [BFK+19]	128	128	251	69	41	1
PKP–192 [BFK+19]	192	192	509	94	54	1
PKP-256 [BFK+19]	256	256	4093	106	47	1

Table 4.1: Parameter sets for different security levels. The security level is estimated based on the attack by Koussa et al. [KMRP19].

may not be as efficient as previously thought for the current PKP security parameters. Koussa, Macario-Rat and Patarin's second contribution is a combination of the ideas of Patarin and Chauvaud [PC93] with the ones by Poupard [Pou97] to obtain a new algorithm to solve PKP, together with a detailed analysis on their time and space complexity.

The main drawback of Koussa's et al. attack is that they use a significant amount of memory, and their implementation may not be efficient in practice. Moreover, all of the published attacks against PKP target the original version of the problem. And even though they all can be adapted to attack the binary PKP, as done by Lampe and Patarin [LP12] for their analysis, it appears that none of the attacks are significantly more efficient in the binary case.

4.3.2 Instantiation

We now present the parameter sets for PKP, in which the security level is estimated based on the best attacks available by Koussa et al. [KMRP19]. Table 4.1 shows these parameter sets for different security levels. The focus of this work is in the parameter sets given in the first two rows, corresponding to the binary PKP. It is important to notice that what we now consider to be the parameter sets BPKP–76 and BPKP–89, originally targeted security levels 79 and 98, respectively. However, these had to be revised after Koussa et al's [KMRP19] attack.

4.4 A novel attack against binary PKP

We are given the public matrices $\mathbf{A}$ and $\mathbf{V}$ and we want to find the secret permutation π such that $\mathbf{AV}_\pi = \mathbf{0}$. Let $\mathcal{C}_\mathbf{A}$ and $\mathcal{C}_\mathbf{K}$ be the binary codes generated by $\mathbf{A}$ and $\mathbf{K}$, respectively, where $\mathbf{K}$ is the left kernel matrix of $\mathbf{V}$. Fix an integer w small enough so that we can build the sets $\mathcal{L}^w_\mathbf{A}$ and $\mathcal{L}^w_\mathbf{K}$ consisting of all the codewords of weight w in $\mathcal{C}_\mathbf{A}$ and $\mathcal{C}_\mathbf{K}$, correspondingly. Notice that, since $\mathbf{AV}_\pi = \mathbf{0}$, then $\mathcal{L}^w_\mathbf{A} \subset \mathcal{L}^w_{\pi(\mathbf{K})} = \{\mathbf{u}_\pi : \mathbf{u} \in \mathcal{L}^w_\mathbf{K}\}$.

This idea gives the following simple algorithm to find the secret permutation π. First find a subset S of $\mathcal{L}^w_\mathbf{K}$, such that, for some permutation τ, $\mathcal{L}^w_\mathbf{A} = \{\mathbf{u}_\tau : \mathbf{u} \in S\}$. Then, test if the corresponding column permutation τ is valid, that is, if $\mathbf{AV}_\tau = \mathbf{0}$. If τ is valid, return it as π. Otherwise, restart the search. Figure 4.1 can be useful for visualizing the relationship

between the two sets of codewords, which is the core of the attack.

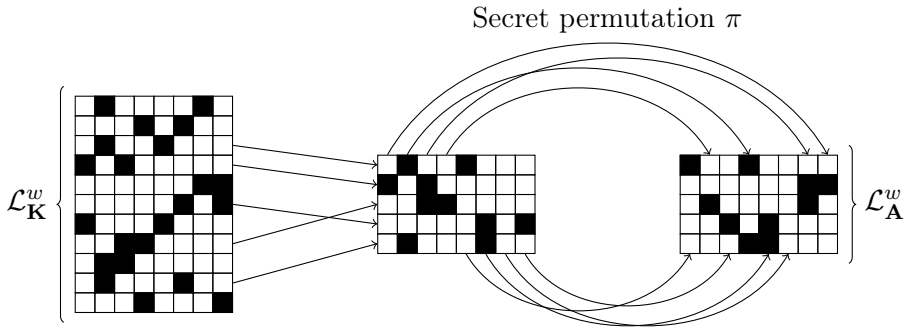

Figure 4.1: Illustration of the relationship between $\mathcal{L}^w_{\mathbf{A}}$ and $\mathcal{L}^w_{\mathbf{K}}$ with respect to the secret column permutation π for codewords of weight $w = 2$. White and black squares represent null and non-null entries, respectively.

Even though it has a rather simple description, we need to carefully deal with the following two problems. The first one is that matching vectors in $\mathcal{L}^w_{\mathbf{A}}$ and a subset of $\mathcal{L}^w_{\mathbf{K}}$ is closely related to the subgraph isomorphism problem, which is NP-hard [GJ79]. The second problem is that, since we are dealing with sparse codewords, there may be a large number of repeated columns in $\mathcal{L}^w_{\mathbf{A}}$. This could potentially make it infeasible to find the secret permutation π because of the combinatorial explosion on the number of possible permutations between columns.

In the following sections, we formally describe the algorithms for the attack against the binary PKP. Then, after this initial exposition, each component of the algorithm is analyzed in Section 4.5.

4.4.1 Searching for codewords of small weight

The problem of finding codewords of small weight is hard in general, with the security of some well known cryptographic schemes, such as McEliece's one [McE78], depend on this problem's hardness. However, in the binary PKP setting, the length n of the codes in question, namely $\mathcal{C}_{\mathbf{A}}$ and $\mathcal{C}_{\mathbf{K}}$, is typically very small, which makes it even possible to use brute force. Using brute force, one has to test exactly if $\binom{n}{w}$ words are elements of each of the codes.

A better approach would be to use specialized algorithms from Coding Theory such as Stern's algorithm [Ste88], which we used in our attack implementation, or its improved variants [FS09, BLP11]. All of these are are well-known probabilistic algorithms that can be used to find low weight codewords in binary codes.

4.4.2 Searching for matchings

Aiming to simplify the description of the attack, we identify sets $\mathcal{L}^w_{\mathbf{A}}$ and $\mathcal{L}^w_{\mathbf{K}}$ as matrices where each row is one vector in the corresponding set. This is arguably a natural

identification when we consider a real implementation of the algorithm in a programming language such as C.

We now focus on the problem of finding a submatrix of $\mathcal{L}^w_{\mathbf{K}}$ which is equal to matrix $\mathcal{L}^w_{\mathbf{A}}$ when its coordinates are permuted by some permutation τ. Notice that, if we let $\mathcal{G}(\mathbf{X})$ be the bipartite graph built using matrix $\mathbf{X}$ as a biadjacency matrix, then this problem is exactly the subgraph isomorphism problem for the bipartite graphs $\mathcal{G}(\mathcal{L}^w_{\mathbf{A}})$ and $\mathcal{G}(\mathcal{L}^w_{\mathbf{K}})$.

Even though subgraph isomorphism is NP-hard [GJ79], for small enough inputs, the problem has been widely studied because of its importance in Pattern Recognition. It is well-known that, for sufficiently small instances, the problem can be solved efficiently using algorithms such as the one by Ullman [Ull76] or the ones from the VF family [SV01, CFSV04]. The main problem with these widely used algorithms is that they use heuristics that make it hard to perform a sound average case complexity analysis for our case. Since such analysis is critical for estimating the concrete security of the scheme, we propose a different algorithm with two remarkable advantages. The first one is that it runs faster than other generic subgraph isomorphism algorithms for our specific case of bipartite graphs. The second is that it is simpler to analyze and give realistic estimates on its performance.

The algorithm we propose is based on a simple depth-first search strategy. In each level α of the search, a node represents a matrix built using a set of α rows of $\mathcal{L}^w_{\mathbf{K}}$ which is equal to the first α rows of $\mathcal{L}^w_{\mathbf{A}}$, when its columns are permuted by some permutation. Whenever a matching is found, the searching algorithm calls a procedure that tries to extract the secret permutation from the matching. In the following sections, we describe each component of the algorithm in more detail.

Signature of a matrix

It is crucial for the subgraph isomorphism algorithms to efficiently determine whether a matrix $\mathbf{S}$ is equal to a submatrix of $\mathcal{L}^w_{\mathbf{A}}$ up to some column permutation. For this task, we can use a function σ such that, if $\sigma(\mathbf{S}_1) = \sigma(\mathbf{S}_2)$, then with high probability $\mathbf{S}_1 = \tau(\mathbf{S}_2)$ for some permutation τ, for any two matrices $\mathbf{S}_1$ and $\mathbf{S}_2$ with equal dimensions.

One easy way to build such a function is to sort the columns of $\mathbf{S}$ using a lexicographical ordering obtaining $\mathbf{S}^{\text{Sorted}}$. Then, the signature of $\mathbf{S}$ is simply $\sigma(\mathbf{S}) = h\left(\mathbf{S}^{\text{Sorted}}\right)$, for some cryptographic hash function h. It is clear, by this construction, that σ is invariant with respect to column permutations.

The problem with sorting is that, since this function will be executed a very large number of times, it can become expensive. One alternative is to use the following approximation $\sigma(\mathbf{S}) = \sum_{\mathbf{c} \text{ column of } \mathbf{S}} h(\mathbf{c})$, for some hash function h.

Precomputation of signatures

This step consists in building the $|\mathcal{L}^w_{\mathbf{A}}| \times |\mathcal{L}^w_{\mathbf{A}}|$ matrix $\mathbf{H}$ containing signatures of submatrices of $\mathcal{L}^w_{\mathbf{A}}$ that are used for pruning the possible child nodes in each level of the search. Let $\mathbf{a}_1, \ldots, \mathbf{a}_{|\mathcal{L}^w_{\mathbf{A}}|}$ be the vectors in $\mathcal{L}^w_{\mathbf{A}}$, and let $\mathbf{L}_j$ denote the matrix formed by the first j

rows of $\mathcal{L}_{\mathbf{A}}^{w}$. Then, we let

$$\mathbf{H}_{i,j} = \begin{cases} \sigma\left(\left[\dfrac{\mathbf{L}_j}{\mathbf{a}_i}\right]\right) & \text{if } i > j, \text{ and} \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

Key recovery algorithm

In this step, the algorithm effectively tries to build a submatrix $\mathbf{S}$ of $\mathcal{L}_{\mathbf{K}}^{w}$ such that $\tau(\mathbf{S}) = \mathcal{L}_{\mathbf{A}}^{w}$, for some column permutation τ. The algorithm is formally described as Algorithm 4.1 but we give a brief description next.

Algorithm 4.1 KeySearch: Key search algorithm using depth-first search.

▷ $\mathbf{A}$ and $\mathbf{V}$: the PKP public parameters
▷ $\mathcal{L}_{\mathbf{A}}^{w}$: a set of codewords in $\mathcal{C}_{\mathbf{A}}$ of weight w
▷ $\mathcal{L}_{\mathbf{K}}^{w}$: the set of all codewords in $\mathcal{C}_{\mathbf{K}}$ of weight w
▷ $\mathbf{H}$: the precomputed matrix of signatures
▷ α: the level in the search tree (initially, $\alpha = 0$)
▷ $\mathbf{S}$: an $\alpha \times n$ matrix (initially, $\mathbf{S}$ is the empty $0 \times n$ matrix)
▷ $\mathcal{P} = \left(P_1, \ldots, P_{|\mathcal{L}_{\mathbf{A}}^{w}|}\right)$: the sets of children (initially, each $P_i = \mathcal{L}_{\mathbf{K}}^{w}$)
procedure KeySearch($\mathbf{A}$, $\mathbf{V}$, $\mathcal{L}_{\mathbf{A}}^{w}$, $\mathcal{L}_{\mathbf{K}}^{w}$, $\mathbf{H}$, α, $\mathbf{S}$, $\mathcal{P}$)
  if $\alpha = |\mathcal{L}_{\mathbf{A}}^{w}|$ **then**
    ▷ Return π: a permutation such that $\mathbf{AV}_{\pi} = \mathbf{0}$ or $\perp$ if none exists
    return ExtractPermutationFromMatching($\mathbf{A}, \mathbf{V}, \mathbf{S}$)
  for $i = \alpha + 1$ to $|\mathcal{L}_{\mathbf{A}}^{w}|$ **do**
    $\hat{P}_i \leftarrow \left\{\mathbf{p} \in P_i : \sigma\left(\left[\dfrac{\mathbf{S}}{\mathbf{p}}\right]\right) = \mathbf{H}_{i,\alpha}\right\}$ ▷ The possible children for each level not yet defined
  $\mathcal{P} \leftarrow \left(P_1, \ldots, P_{\alpha}, \hat{P}_{\alpha+1}, \ldots, \hat{P}_{|\mathcal{L}_{\mathbf{A}}^{w}|}\right)$
  for each $\mathbf{p}$ in $\hat{P}_{\alpha+1}$ **do**
    Update $\mathbf{S}$ by inserting $\mathbf{p}$ as its last row
    $\pi \leftarrow$ KeySearch($\mathbf{A}, \mathbf{V}, \mathcal{L}_{\mathbf{A}}^{w}, \mathcal{L}_{\mathbf{K}}^{w}, \mathbf{H}, \alpha + 1, \mathbf{S}, \mathcal{P}$) ▷ Recursive call
    if $\pi \neq \perp$ **then**
      return π
    Update $\mathbf{S}$ by removing its last row $\mathbf{p}$
  return $\perp$

The search starts at level $\alpha = 0$, with $\mathbf{S}$ being a $0 \times n$ empty matrix. At each level α in the search tree, the algorithm runs a pruning procedure, that updates the lists of possible vectors for each level greater than α using the precomputed matrix of signatures $\mathbf{H}$. This ensures that the main invariant of the recursive algorithm is that, at level α, the algorithm holds an $\alpha \times n$ submatrix $\mathbf{S}$ of $\mathcal{L}_{\mathbf{K}}^{w}$ which is equal to the matrix formed by the first α rows of $\mathcal{L}_{\mathbf{A}}^{w}$, up to some column permutation. The search proceeds by selecting a vector from set $P_{\alpha+1} \subset \mathcal{L}_{\mathbf{K}}^{w}$ of vectors which can be safely added to the next level without breaking the invariant. Each time the algorithm successfully gets to a leaf, that is, it adds a vector to

level $\alpha = |\mathcal{L}_{\mathbf{A}}^{w}|$, then a full matching $\mathbf{S}$ is found and the procedure that tries to extract the permutation π from matching $\mathbf{S}$ is executed. If the permutation π is successfully extracted, then π is returned. Otherwise, the depth-first search proceeds.

The procedure for extracting the secret permutation from the matching, if possible, is described in the next section.

4.4.3 Extracting permutations from matchings

After each each matching found in the previous step, we are given a matrix $\mathbf{S}$ such that $\tau(\mathbf{S}) = \mathcal{L}_{\mathbf{A}}^{w}$ for at least one permutation of columns τ. In this section we describe how to efficiently extract the secret permutation π from this matching, if possible.

We first consider the brute force solution. Let T be the set of permutations that match equal columns in $\mathbf{S}$ and $\mathcal{L}_{\mathbf{A}}^{w}$. Then, we can just test, for each permutation τ in T, if $\mathbf{A}\mathbf{V}_{\tau} = \mathbf{0}$. If one such τ is found, then the algorithm returns $\pi \leftarrow \tau$. Suppose that there are β unique columns $\mathbf{c}_1, \ldots, \mathbf{c}_\beta$ of matrix $\mathcal{L}_{\mathbf{A}}^{w}$, and let c_i denote the number of times column $\mathbf{c}_i$ appears in $\mathcal{L}_{\mathbf{A}}^{w}$. This implies that the number of candidate permutations is given by $|T| = \prod_{i=1}^{\beta} (c_i!)$. The brute force approach may be efficient when $\mathbf{S}$ has a large number of unique columns. However, due to the combinatorial nature of this problem, even a small increase in the number of equal columns can make the algorithm very inefficient.

To reduce the number of permutations to test we can use the fact that $\dim(\ker \mathbf{A}) = n-m$. Therefore, there are $n-m$ rows of $\mathbf{V}_{\pi}$ which, together with the m equations defined by $\mathbf{A}$, completely determine the other m rows of $\mathbf{V}_{\pi}$. Intuitively, this means we can focus on partial permutations in T corresponding to these $n-m$ indexes.

More formally, let I_1 and I_2 be a partition of the set of possible n indexes such that $|I_1| = m$ and the $m \times m$ matrix $\mathbf{A}_1$ built using the columns from $\mathbf{A}$ whose indexes are in I_1 is invertible. Similarly, let $\mathbf{A}_2$ be the $m \times (n-m)$ matrix whose columns are taken from $\mathbf{A}$, but with indexes in I_2. Let ϕ be the permutation of n elements such that $\phi(\mathbf{A}) = [\mathbf{A}_1|\mathbf{A}_2]$, and define as $\mathbf{U}_1$ and $\mathbf{U}_2$ the matrices such that $\phi\left((\mathbf{V}_{\pi})^{\top}\right) = (\mathbf{V}_{\pi\phi})^{\top} = \left[\, \mathbf{U}_1^{\top} \,|\, \mathbf{U}_2^{\top} \,\right]$. Then, we have

$$\mathbf{A}\mathbf{V}_{\pi} = \phi(\mathbf{A})\mathbf{V}_{\pi\phi} = [\mathbf{A}_1|\mathbf{A}_2] \left[\frac{\mathbf{U}_1}{\mathbf{U}_2}\right] = \mathbf{A}_1\mathbf{U}_1 + \mathbf{A}_2\mathbf{U}_2 = \mathbf{0},$$

which implies that $\mathbf{U}_1 = \left(\mathbf{A}_1^{-1}\mathbf{A}_2\right)\mathbf{U}_2$.

Therefore, one can reduce the number of permutations in T to test by using the following procedure. Let I be a sequence of n column indexes sorted, in decreasing order, by the number of times in which the corresponding column of matrix $\mathcal{L}_{\mathbf{A}}^{w}$ occurs in this same matrix.[21] Now let I_1 to be composed by the first m indexes in I whose corresponding columns of $\mathbf{A}$ are linearly independent, and let $I_2 = I - I_1 = \{i_1, \ldots, i_{n-m}\}$. Consider the

[21] The reason why it is interesting to sort the indexes in this way is explained in the last paragraph of this section.

set

$$\mathcal{J} = \left\{ (j_1, \ldots, j_{n-m}) : (\mathbf{S})^{j_k} = (\mathcal{L}_{\mathbf{A}}^w)^{i_k} \text{ for all } k = 1, \ldots, n-m \right\},$$

where $(\mathbf{X})^y$ denotes the y–th column of matrix $\mathbf{X}$. Intuitively, set $\mathcal{J}$ captures the parts of the permutations in T corresponding only to the $n-m$ indexes in I_2, and therefore $|\mathcal{J}|$ may be much smaller than $|T|$. For each sequence J of $\mathcal{J}$, we let $\mathbf{U}_2$ be the $(n-r) \times \ell$ matrix built from rows of $\mathbf{V}$ whose indexes are in J. For each of these possible values of $\mathbf{U}_2$, we compute the matrix $\mathbf{U}_1 = \left(\mathbf{A}_1^{-1}\mathbf{A}_2\right)\mathbf{U}_2$, and test if $\left[\begin{array}{c}\mathbf{U}_1\\\hline\mathbf{U}_2\end{array}\right]$ corresponds to a permutation of the rows of $\mathbf{V}$. If this is indeed the case, then the secret matrix $\mathbf{V}_\pi$ is simply $\mathbf{V}_\pi = \left[\begin{array}{c}\mathbf{U}_1\\\hline\mathbf{U}_2\end{array}\right]_{\phi^{-1}}$.

It is important to notice that, since we want to make $|\mathcal{J}|$ as low as possible, we sorted the set of indexes I so that, when defining I_1 and I_2, the columns of $\mathcal{L}_{\mathbf{A}}^w$ whose indexes are in I_2 tend to appear a lower number of times. In Section 4.5.3, we show how to estimate the size of $\mathcal{J}$.

4.5 Concrete analysis of the attack

In this section we estimate the attack complexity. We begin by analyzing, in the first three subsections, the work factor of the three components of the attack algorithm: building sets $\mathcal{L}_{\mathbf{A}}^w$ and $\mathcal{L}_{\mathbf{K}}^w$, matching the low weight vectors in these sets, and extracting the secret permutation from matchings. Then, we put these components together to give the complexity of the attack in Section 4.5.4. Finally, in Section 4.5.5, we show the performance of the attack in practice.

The work factor of attacks against PKP is typically stated in number of matrix-vector products, as it is the basic operation to test if a vector is in the kernel of a matrix. Even though binary PKP uses two matrices, we can see the rows of $\mathbf{V}$ as elements of $\mathbb{F}_{2^\ell}$ and, since ℓ is typically small, then the product $\mathbf{AV}$ can be seen as a matrix-vector multiplication where sum is replaced by a XOR.

4.5.1 Searching for codewords of small weight

Let us analyze the first step of the attack: the construction of sets $\mathcal{L}_{\mathbf{A}}^w$ and $\mathcal{L}_{\mathbf{K}}^w$. Each of these sets can be computed by searching exhaustively the whole set of $\binom{n}{w}$ possible vectors of n bits of weight w, and testing if they belong to $\mathcal{C}_{\mathbf{A}}$ and $\mathcal{C}_{\mathbf{K}}$. However, as we pointed in Section 4.4.1, we can do a lot better by using Stern's [Ste88] algorithm. Consider a random $[n,k]$–linear code generated by matrix $\mathbf{G}$. Given parameters (p,q), Stern's algorithm first permutes the columns of $\mathbf{G}$ hoping to obtain a matrix $\hat{\mathbf{G}} = \phi(\mathbf{G})$, called a good permutation, for which there is a linear combination of its rows that has the form $\mathbf{c} = [\mathbf{c}_1|\mathbf{c}_2|\mathbf{c}_3|\mathbf{c}_4]$, such that $\mathrm{w}(\mathbf{c}_1) = \mathrm{w}(\mathbf{c}_2) = p$, component $\mathbf{c}_3$ is the zero vector of length q, and $\mathbf{c}_4$ has weight $\mathrm{w}(\mathbf{c}_4) = w - 2p$.

When such conditions are met, Stern's algorithm finds a vector $\mathbf{c}$ with such properties, which can then be permuted to give a vector $\mathbf{c}_{\phi^{-1}}$ of weight w in the code generated by $\mathbf{G}$. To compute the work factor of Stern's algorithm then, we have to take into account the average number of iterations until it chooses a good permutation $\hat{\mathbf{G}}$ and the average number of operations performed by the algorithm each time. Considering Finiasz and Sendrier's [FS09] approximation, which takes parameter $q \approx \log\binom{k/2}{p}$, the work factor of Stern's algorithm, considering the number of binary operations, until it gives us a random codeword of weight w in a random $[n,k]$–linear code is

$$\mathbf{BinOpsWF}^{(n,k,w)}_{\text{Stern}} \approx \min_p \frac{2q\binom{n}{w}}{\binom{n-k-q}{w-2p}\binom{k/2}{p}}.$$

Each time Stern's algorithm runs successfully, it finds a random codeword of weight w. Therefore we can model the expected number of iterations until all codewords are found as an instance of the coupon collector problem. Let us consider the time to build $\mathcal{L}^w_{\mathbf{A}}$. Each low weight vectors is modeled as a coupon, and we need to collect all $\ell_{\mathbf{A}}$ of them. Let C be the random variable that counts the number of low weight vectors we need to find before obtaining $\ell_{\mathbf{A}}$ different vectors. Then, it is well known that $\mathbb{E}(C) = \Theta(\ell_{\mathbf{A}} \log \ell_{\mathbf{A}})$. Furthermore, the upper tail estimate for the coupon collector problem ensures that

$$\Pr(C \geq \gamma_{\mathbf{A}} \ell_{\mathbf{A}} \log \ell_{\mathbf{A}}) \leq \ell_{\mathbf{A}}^{-\gamma_{\mathbf{A}}+1}.$$

Let $\mathbf{WF}_{\mathcal{L}^w_{\mathbf{A}}}$ be the work factor of building the set $\mathcal{L}^w_{\mathbf{A}}$, counted in number of binary matrix-vector multiplications. Since $\dim \mathbf{A} = m$, we can get an upper bound on $\mathbf{WF}_{\mathcal{L}^w_{\mathbf{A}}}$ as

$$\mathbf{WF}^{(n,m,w,\ell_{\mathbf{A}})}_{\mathcal{L}^w_{\mathbf{A}}} \leq \mathbf{BinOpsWF}^{(n,m,w,\ell_{\mathbf{A}})}_{\mathcal{L}^w_{\mathbf{A}}} = \gamma_{\mathbf{A}} (\ell_{\mathbf{A}} \log \ell_{\mathbf{A}}) \mathbf{BinOpsWF}^{(n,m,w)}_{\text{Stern}},$$

where $\gamma_{\mathbf{A}} > 1$ is chosen so that $\ell_{\mathbf{A}}^{-\gamma_{\mathbf{A}}+1}$ gives a small error probability.

Now we want do the same thing for the construction of $\mathcal{L}^w_{\mathbf{K}}$. Let $\ell_{\mathbf{K}} = |\mathcal{L}^w_{\mathbf{K}}|$, and let us estimate $\ell_{\mathbf{K}}$. As usual in coding theory, to count elements of a given weight, we approximate the number of elements of weight w in a random code as a binomial distribution. Thus, out of the $\binom{n}{w}$ possible vectors of weight w, we expect that a fraction of $2^{\dim \mathbf{K}}/2^n$ belong to $\mathcal{C}_{\mathbf{K}}$. Since $\mathbf{K}$ is the left kernel matrix of $\mathbf{V}$, then $\dim \mathbf{K} = (n - \dim \mathbf{V}) = (n - \ell)$, and we can approximate the expected value of $\ell_{\mathbf{K}}$ as

$$\hat{\ell}_{\mathbf{K}} = \mathbb{E}(\ell_{\mathbf{K}}) \approx \frac{2^{n-l}}{2^n}\binom{n}{w} = 2^{-l}\binom{n}{w}. \tag{4.2}$$

Therefore, for some factor $\gamma_{\mathbf{K}} > 1$ we can define an upper bound on the work factor of building $\mathcal{L}^w_{\mathbf{K}}$ as

$$\mathbf{WF}^{(n,\ell,w,\ell_{\mathbf{A}})}_{\mathcal{L}^w_{\mathbf{K}}} \leq \gamma_{\mathbf{K}} \left(\hat{\ell}_{\mathbf{K}} \log \hat{\ell}_{\mathbf{K}}\right) \mathbf{BinOpsWF}^{(n,n-\ell,w)}_{\text{Stern}}.$$

Notice that if $\mathcal{L}_{\mathbf{K}}^{w}$ does not contain the permutations of all vectors of $\mathcal{L}_{\mathbf{A}}^{w}$, then the search will fail. Thus, factor $\gamma_{\mathbf{K}}$ must be chosen conservatively, but since $\hat{\ell}_{\mathbf{K}}$ is typically very large, the probability $\hat{\ell}_{\mathbf{K}}^{(1-\gamma_{\mathbf{K}})}$ of not collecting all vectors can be made negligible even for relatively small $\gamma_{\mathbf{K}}$.

4.5.2 Searching for matchings

In this section, we evaluate the number of paths that will be tested by the subgraph isomorphism algorithm. For this evaluation, we need to estimate the number of possible child nodes in each level.

Consider the case when the search is holding matrix $\mathbf{S}$ at level α. We want to estimate the size of set $\hat{P}_{\alpha+1}$ of possible rows to add to $\mathbf{S}$ in the next level of the search. In other words, we want to compute the number of vectors that survive the filter imposed by the line 13 of Algorithm 4.1. The first thing to notice is that the result of the filtering is exactly the same if we filter from $\mathbf{p} \in \mathcal{L}_{\mathbf{K}}^{w}$ instead of $\mathbf{p} \in P_i$, that is

$$\hat{P}_i = \left\{ \mathbf{p} \in P_i : \sigma\left(\left[\frac{\mathbf{S}}{\mathbf{p}}\right]\right) = \mathbf{H}_{i,\alpha} \right\} = \left\{ \mathbf{p} \in \mathcal{L}_{\mathbf{K}}^{w} : \sigma\left(\left[\frac{\mathbf{S}}{\mathbf{p}}\right]\right) = \mathbf{H}_{i,\alpha} \right\}.$$

The reason why the algorithm keeps updating the list $\mathcal{P} = \left(P_1, \ldots, P_{|\mathcal{L}_{\mathbf{A}}^{w}|}\right)$ of possible vectors in all levels is solely for efficiency. Without it, the filtering would be very inefficient for nodes in lower levels down the search because it would have to run, every time, through set $\mathcal{L}_{\mathbf{K}}^{w}$, which may be very large.

Let $\mathbf{L}_\alpha$ be the matrix formed by the first α rows of $\mathcal{L}_{\mathbf{A}}^{w}$, and let $\mathbf{r}$ be the $(\alpha+1)$–th row of $\mathcal{L}_{\mathbf{A}}^{w}$. Now, using the definition of $\mathbf{H}_{i,\alpha}$, we want to estimate how many vectors $\mathbf{p}$ in $\mathcal{L}_{\mathbf{K}}^{w}$ satisfy $\sigma\left(\left[\frac{\mathbf{S}}{\mathbf{p}}\right]\right) = \sigma\left(\left[\frac{\mathbf{L}_\alpha}{\mathbf{r}}\right]\right)$.

One problem that makes estimating the number of child nodes difficult is that, since vectors in $\mathcal{L}_{\mathbf{K}}^{w}$ are low-weight codewords of a fixed linear code, the vectors in $\mathcal{L}_{\mathbf{K}}^{w}$ are not independently distributed. This is a common problem when analyzing bounds on weight distribution in coding theory, and as usual in the field, we overcome this problem by assuming that the set $\mathcal{L}_{\mathbf{K}}^{w}$ consists of vectors chosen uniformly at random over the vectors of length n and weight w.

Now, under our model, let us fix $\mathbf{L}_\alpha$ and estimate the probability $q_{\alpha+1}(\mathbf{L}_\alpha)$ that vector $\hat{\mathbf{p}}$ of $\mathcal{L}_{\mathbf{K}}^{w}$ is a possible child node in $\hat{P}_{\alpha+1}$. Because of the way that the algorithm builds $\mathbf{S}$, its columns are the same as the ones of $\mathbf{L}_\alpha$, up to some permutation, and therefore

$$\begin{aligned} q_{\alpha+1}(\mathbf{L}_\alpha) &= \Pr\left(\sigma\left(\left[\frac{\mathbf{S}}{\hat{\mathbf{p}}}\right]\right) = \sigma\left(\left[\frac{\mathbf{L}_\alpha}{\mathbf{r}}\right]\right)\right) \\ &= \Pr\left(\sigma\left(\left[\frac{\mathbf{L}_\alpha}{\mathbf{p}}\right]\right) = \sigma\left(\left[\frac{\mathbf{L}_\alpha}{\mathbf{r}}\right]\right)\right), \end{aligned}$$

where $\mathbf{p}$ is a random n–bit vector of weight w.

The signatures of the two matrices will be the same if the columns above the non-null entries of $\mathbf{p}$ and $\mathbf{r}$ are equal, up to some permutation. Therefore, $q_{\alpha+1}(\mathbf{L}_\alpha)$ is simply the probability that two subsets of w columns drawn from $\mathbf{L}_\alpha$ are the same, up to some permutation. In the simple case when all columns of $\mathbf{L}_\alpha$ are unique, then $q_{\alpha+1}(\mathbf{L}_\alpha) = 1/\binom{n}{w}$. However, in general, $\mathbf{L}_\alpha$ may have non-unique columns, which occur with high probability for small values of α, since $\mathbf{L}_\alpha$ is sparse.

Let $\mathbf{R}$ be the $\alpha \times w$ matrix built by taking columns of $\mathbf{L}_\alpha$ whose indexes are in $\mathrm{supp}\,(\mathbf{r})$. Define two counting functions N and $N_{\mathbf{R}}$ that, given a column $\mathbf{c}$, output the number of times column $\mathbf{c}$ appears in matrices $\mathbf{L}_\alpha$ and $\mathbf{R}$, respectively. For each column $\mathbf{c}$, which should appear $N_{\mathbf{R}}(\mathbf{c})$ times in the columns above the non-null entries of $\mathbf{p}$, there are $\binom{N(\mathbf{c})}{N_{\mathbf{R}}(\mathbf{c})}$ ways in which different column indexes of $\mathbf{R}$ can be chosen. Therefore

$$q_{\alpha+1}(\mathbf{L}_\alpha) = \frac{1}{\binom{n}{w}} \prod_{\mathbf{c} \in \mathbb{F}_2^\alpha} \binom{N(\mathbf{c})}{N_{\mathbf{R}}(\mathbf{c})}.$$

To estimate the average attack performance, we want to compute the expected value $\overline{q}_{\alpha+1} = \mathbb{E}\,(q_{\alpha+1}(\mathbf{L}_\alpha))$ when $\mathbf{L}_\alpha$ is obtained from a randomly generated key. This value can be easily estimated using simulations by sampling $\mathbf{L}_\alpha$ from the set of $\alpha \times n$ matrices in which each row has weight w. However, to give an analytic approximation, we face the problem of computing the expected value of the binomial coefficients over the random variables $N(\mathbf{c})$ and $N_{\mathbf{R}}(\mathbf{c})$ for each possible column $\mathbf{c}$. To deal with this problem, we use of the following rough approximation

$$\overline{q}_{\alpha+1} \approx \frac{1}{\binom{n}{w}} \prod_{\mathbf{c} \in \mathbb{F}_2^\alpha} \binom{\mathbb{E}\,(N(\mathbf{c}))}{\mathbb{E}\,(N_{\mathbf{R}}(\mathbf{c}))}.$$

To compute the expected values $\mathbb{E}\,(N(\mathbf{c}))$ and $\mathbb{E}\,(N_{\mathbf{R}}(\mathbf{c}))$, we consider $\mathbf{L}_\alpha$ as a random sparse matrix of density w/n as an approximation of the real case where each of its rows have a fixed weight w. Under this model, the probability that a random column of $\mathbf{L}_\alpha$ is equal to $\mathbf{c}$ depends only on its the weight $k = \mathrm{w}\,(\mathbf{c})$ and and the number α of rows in $\mathbf{L}_\alpha$. This probability is given by

$$p\,(k, \alpha) = \left(\frac{w}{n}\right)^k \left(1 - \frac{w}{n}\right)^{\alpha - k}. \tag{4.3}$$

Thus both $N(\mathbf{c})$ and $N_{\mathbf{R}}(\mathbf{c})$ follow binomial distributions with parameters $(n, p(\mathrm{w}\,(\mathbf{c})\,, \alpha))$

and $(w, p(\mathrm{w}(\mathbf{c}), \alpha))$, respectively. Therefore[22]

$$\begin{aligned}\overline{q}_{\alpha+1} &\approx \frac{1}{\binom{n}{w}} \prod_{\mathbf{c}\in\mathbb{F}_2^{\alpha}} \binom{np(\mathrm{w}(\mathbf{c}),\alpha)}{wp(\mathrm{w}(\mathbf{c}),\alpha)} \\ &\approx \frac{1}{\binom{n}{w}} \prod_{k=0}^{\alpha} \binom{np(k,\alpha)}{wp(k,\alpha)}^{\binom{\alpha}{k}}.\end{aligned}$$

One can then use this analytic approximation or simulations for $\overline{q}_\alpha$ to obtain the number of possible nodes in each level as $\left|\hat{P}_\alpha\right| = \overline{q}_\alpha \hat{\ell}_{\mathbf{K}}$, where $\hat{\ell}_{\mathbf{K}} = \mathbb{E}\left(|\mathcal{L}_{\mathbf{K}}^{w}|\right)$ is given approximately by Equation 4.2. Figure 4.2 shows how the analytic approximation and the value obtained by simulations compare with what is observed during a real attack. We can see that simulations can accurately be used to estimate $\overline{q}_\alpha$ and that the analytic estimate tends to overestimate the number of possible nodes in each level.

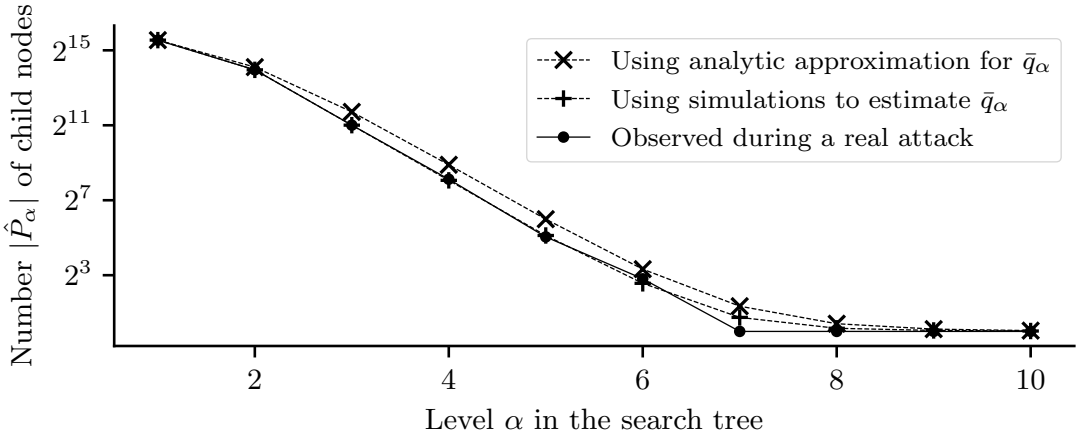

Figure 4.2: Comparison of estimates on the average number of possible vectors to add in each level of the search. The attack parameters ($w = 8, \ell_{\mathbf{A}} = 10$) were used against the BPKP–76 parameter set.

The work factor of the search procedure, denoted by $\mathbf{WF}_{\text{SEARCH}}$, consists of the expected number of possible paths, which is given by

$$\mathbf{WF}_{\text{SEARCH}}^{(n,w,\ell_{\mathbf{A}})} = \prod_{\alpha=1}^{\ell_{\mathbf{A}}} \left|\hat{P}_\alpha\right| \approx \left(\hat{\ell}_{\mathbf{K}}\right)^{\ell_{\mathbf{A}}} \prod_{\alpha=1}^{\ell_{\mathbf{A}}} \overline{q}_\alpha.$$

4.5.3 Extracting permutations from matchings

We now analyze the complexity of the procedure that tries to extract the secret permutation after a matching is found. The main quantity we need to estimate is the number of permutations that the procedure needs to test each time it is called. Formally, we need to estimate the average size of set $\mathcal{J}$ for each parameter set (n, m, ℓ) when the scheme is

[22] Recall, from Section 4.2, that binomials are defined over non-negative real numbers to allow our approximations.

attacked with attack parameters $(w, \ell_{\mathbf{A}})$.

Let I_1 and $I_2 = \{i_1, \ldots, i_{n-m}\}$ be the sets constructed from $\mathcal{L}^w_{\mathbf{A}}$ and $\mathbf{A}$ as described in Section 4.4.3. The first thing to notice is that $|\mathcal{J}|$ can be computed directly from matrix $\mathcal{L}^w_{\mathbf{A}}$, that is, it does not depend on a each $\mathbf{S}$. This is a consequence of the fact that, by construction, $\mathbf{S} = \tau(\mathcal{L}^w_{\mathbf{A}})$, for some column permutation τ. Formally, what we mean is that, since[23]

$$\begin{aligned}\mathcal{J} &= \left\{(j_1, \ldots, j_{n-m}) : (\tau(\mathcal{L}^w_{\mathbf{A}}))^{j_k} = (\mathcal{L}^w_{\mathbf{A}})^{i_k} \text{ for all } k = 1, \ldots, n-m\right\} \\ &= \left\{(j_1, \ldots, j_{n-m})_\tau : (\mathcal{L}^w_{\mathbf{A}})^{j_k} = (\mathcal{L}^w_{\mathbf{A}})^{i_k} \text{ for all } k = 1, \ldots, n-m\right\},\end{aligned}$$

then $|\mathcal{J}| = \left|\left\{(j_1, \ldots, j_{n-m}) : (\mathcal{L}^w_{\mathbf{A}})^{j_k} = (\mathcal{L}^w_{\mathbf{A}})^{i_k} \text{ for all } k = 1, \ldots, n-m\right\}\right|$, which does not depend on τ.

Thus we can model $|\mathcal{J}|$ as the number of arrangements of $n-m$ different balls, which may come from different boxes, under the restriction that each box will be sampled a fixed number of times. In this analogy, each box represents a set of indexes that correspond to equal columns in $\mathcal{L}^w_{\mathbf{A}}$. More formally, let $\mathbf{L}_2$ be the $\ell_{\mathbf{A}} \times (n-m)$ matrix formed by taking columns of $\mathcal{L}^w_{\mathbf{A}}$ whose indexes are in I_2. Define two counting functions N and N_2 that, given a column $\mathbf{c}$, output the number of times column $\mathbf{c}$ appears in matrices $\mathcal{L}^w_{\mathbf{A}}$ and $\mathbf{L}_2$, respectively. Then, we have

$$|\mathcal{J}| = \prod_{\mathbf{c} \in \mathcal{C}_2} \frac{N(\mathbf{c})!}{(N(\mathbf{c}) - N_2(\mathbf{c}))!}.$$

Now, let us consider the expected value of $\mathcal{J}$ when $\mathbf{A}$ is a random matrix such that $\mathcal{L}^w_{\mathbf{A}}$ contains $\ell_{\mathbf{A}}$ vectors of weight w. This number can easily be estimated by simulations, which perfectly correspond to what is observed in a real attack since, up to this point no simplification has been made. Furthermore, we can also give an analytic estimate using the very same ideas from the previous section. First we approximate this case by modeling $\mathcal{L}^w_{\mathbf{A}}$ as a random $\ell_{\mathbf{A}} \times n$ sparse matrix with density w/n, and let $p(k, \ell_{\mathbf{A}})$ denote the probability that a given column of $\mathcal{L}^w_{\mathbf{A}}$ is equal to a fixed column of weight k and height $\ell_{\mathbf{A}}$, as defined by Equation 4.3. Then, the rough approximation on $\mathbb{E}(|\mathcal{J}|)$ is given by

$$\begin{aligned}\mathbb{E}(|\mathcal{J}|) &\approx \prod_{\mathbf{c} \in \mathbb{F}_2^{\ell_{\mathbf{A}}}} \frac{\mathbb{E}(N(\mathbf{c}))!}{(\mathbb{E}(N(\mathbf{c})) - \mathbb{E}(N_2(\mathbf{c})))!} \\ &= \prod_{k=0}^{\ell_{\mathbf{A}}} \left(\frac{(np(k, \ell_{\mathbf{A}}))!}{(np(k, \ell_{\mathbf{A}}) - (n-m)p(k, \ell_{\mathbf{A}}))!}\right)^{\binom{\ell_{\mathbf{A}}}{k}} \\ &= \prod_{k=0}^{\ell_{\mathbf{A}}} \left(\frac{(np(k, \ell_{\mathbf{A}}))!}{(mp(k, \ell_{\mathbf{A}}))!}\right)^{\binom{\ell_{\mathbf{A}}}{k}}.\end{aligned}$$

Figure 4.3 shows how $|\mathcal{J}|$ rapidly decreases as larger values of $\ell_{\mathbf{A}}$ are used. It also pro-

[23] Recall that $(\mathbf{X})^i$ denotes the i–th column of matrix $\mathbf{X}$.

vides a comparison between our analytic estimate on $\mathbb{E}(|\mathcal{J}|)$ and the observed values in our simulations. Notice that, for small values of $\ell_{\mathbf{A}}$, the analytic estimate tends to overestimate the real values of $|\mathcal{J}|$, but for sufficiently large $\ell_{\mathbf{A}}$, the estimate converges to the observed values. Now, since each sequence $\mathbb{E}(|J|)$ needs one matrix multiplication to be tested, we define the work factor of the permutation extraction procedure, as $\mathbf{WF}_{\text{Perms}}^{(n,m,w,\ell_{\mathbf{A}})} = \mathbb{E}(|J|)$.

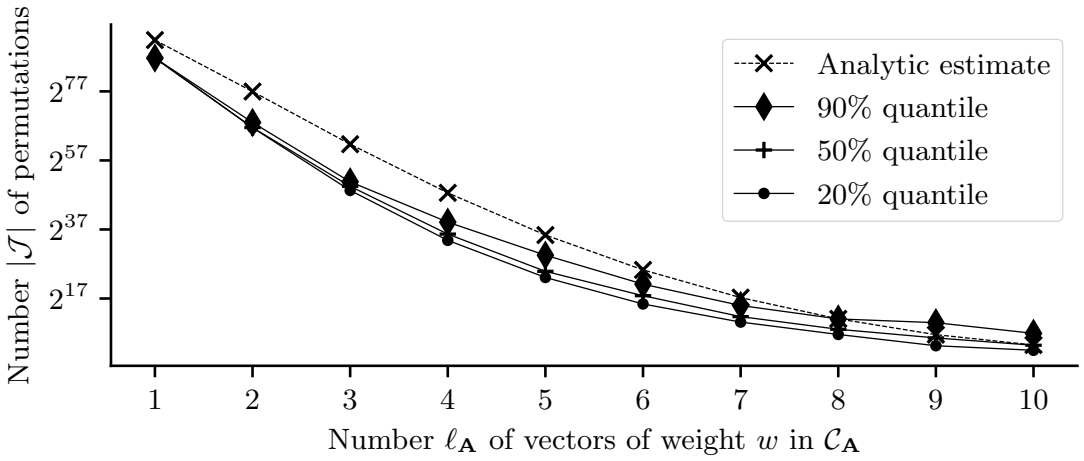

Figure 4.3: The number of permutations to test after each matching considering the BPKP–76 parameter set. The attack parameter $w = 8$ was fixed, and the simulations were run for increasing values of parameter $\ell_{\mathbf{A}}$.

4.5.4 Attack Complexity

This section builds upon the three previous sections to explicitly state the attack complexity and the fraction of keys that can be attacked for different attack parameters $(w, \ell_{\mathbf{A}})$.

The full complexity of the attack is given by the following lemma.

Lemma 4.5.1. *Let (n, m, ℓ) be a binary PKP parameter set. Then, the work factor of the attack with parameters $(w, \ell_{\mathbf{A}})$ is given as*

$$\mathbf{WF}_{\text{Attack}}^{(n,m,\ell,w,\ell_{\mathbf{A}})} = \mathbf{WF}_{\text{LowWeightSets}}^{(n,m,\ell,w,\ell_{\mathbf{A}})} + \left(\mathbf{WF}_{\text{Search}}^{(n,w,\ell_{\mathbf{A}})}\right)\left(\mathbf{WF}_{\text{Perms}}^{(n,m,w,\ell_{\mathbf{A}})}\right).$$

Proof. The complexity of the attack is given by summing the costs of building the sets of vectors of small weight $\mathcal{L}_{\mathbf{A}}^{w}$ and $\mathcal{L}_{\mathbf{K}}^{w}$, and the complexity of the key recovery algorithm. The cost of the key recovery algorithm is computed as follows. Remember that, for each path, from the root to one leaf, the number of permutations we have to test is given as $\mathbf{WF}_{\text{Perms}}^{(n,m,w,\ell_{\mathbf{A}})}$. Since the average number of paths is $\mathbf{WF}_{\text{Search}}^{(n,w,\ell_{\mathbf{A}})}$, the complexity of the key recovery algorithm is simply the product $\left(\mathbf{WF}_{\text{Search}}^{(n,w,\ell_{\mathbf{A}})}\right)\left(\mathbf{WF}_{\text{Perms}}^{(n,m,w,\ell_{\mathbf{A}})}\right)$. □

Figure 4.4 shows how $\mathbf{WF}_{\text{Attack}}^{(n,m,\ell,w,\ell_{\mathbf{A}})}$ varies with respect to the attack parameters used, when attacking BPKP–76 parameter set. To estimate the work factor of the attack, we used

simulations[24] for $\mathbf{WF}_{\text{SEARCH}}^{(n,w,\ell_{\mathbf{A}})}$ and analytic estimation for $\mathbf{WF}_{\text{PERMS}}^{(n,m,w,\ell_{\mathbf{A}})}$. Notice how, as $\ell_{\mathbf{A}}$ gets larger, the work factor stabilizes. This happens because the number of permutations to test gets closer to 1. Furthermore, it is clear that when w is smaller, the attack is more efficient, which happens because, in this case, $|\mathcal{L}_{\mathbf{K}}^{w}|$ is smaller, which makes the search much faster. The problem however, is that the attack parameters for which the attack is most efficient occur with lower probability, as we elaborate next.

Lemma 4.5.1 does not say anything about the fraction of keys that one can attack using parameters $(w, \ell_{\mathbf{A}})$. To compute this fraction, we have to take into account the probability that a public matrix $\mathbf{A}$, selected at random, generates a code with at least $\ell_{\mathbf{A}}$ codewords of weight w. This is considered in the following lemma.

Lemma 4.5.2. *Let (n, m, ℓ) be a binary PKP parameter set. Then, the fraction of keys against which the attack is effective when using parameters $(w, \ell_{\mathbf{A}})$ is given as*

$$\mathbf{KF}_{\text{ATTACK}}^{n,m,\ell,w,\ell_{\mathbf{A}}} \approx 1 - e^{-\lambda} \sum_{k=0}^{\ell_{\mathbf{A}}-1} \frac{\lambda^k}{k!}, \tag{4.4}$$

where $\lambda = \binom{n}{w} 2^{m-n}$.

Proof. Take a random matrix $\mathbf{A}$, generated with parameters (n, m, ℓ). Let L_w be the random variable that represents the number of vectors of weight w in the code generated by matrix $\mathbf{A}$. Since the code generated by $\mathbf{A}$ is a random code, we can approximate L_w by a binomial distribution which samples $\binom{n}{w}$ vectors and each one of them is in the code with probability 2^{m-n}.

The probability that $L_w \geq \ell_{\mathbf{A}}$ would be then simply $\left(1 - \sum_{k=0}^{\ell_{\mathbf{A}}-1} \Pr(L_w = k)\right)$. However, the probability mass function of the binomial can be costly to compute for some values of k, since N may be very large, and $N - k$ appears as an exponent. But, for large N and small probability 2^{m-n}, the binomial may be approximated as a Poisson distribution with parameter $\lambda = \binom{n}{w} 2^{m-n}$. Then, the approximation given as Equation 4.4 is easily achieved by considering the cumulative distribution function of the Poisson distribution, instead of the binomial. □

Figure 4.5 shows the effect of parameters $(w, \ell_{\mathbf{A}})$ in the fraction of keys that we can attack. The first thing to notice is that large $\ell_{\mathbf{A}}$ and small w tend to occur with smaller probability. Now we can combine both Figures 4.4 and 4.5 to understand the power of the attack. For example, considering parameters $(w = 7, \ell_{\mathbf{A}} = 10)$, we can attack about 1 in each 150.000 keys of BPKP–76 with less than 2^{55} operations, and about 100% of all keys can be recovered using 2^{62} operations.

[24] Even though the analytic approach is useful to estimate the number of nodes in each level, the errors would accumulate exponentially in the product necessary to compute the work factor of the search.

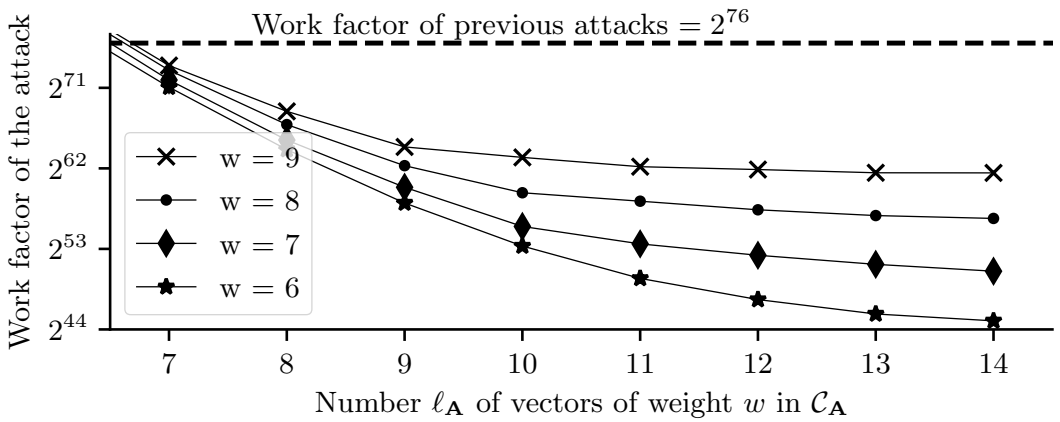

Figure 4.4: Work factor of the attack against BPKP–76 parameter set using different attack parameters $(w, \ell_{\mathbf{A}})$.

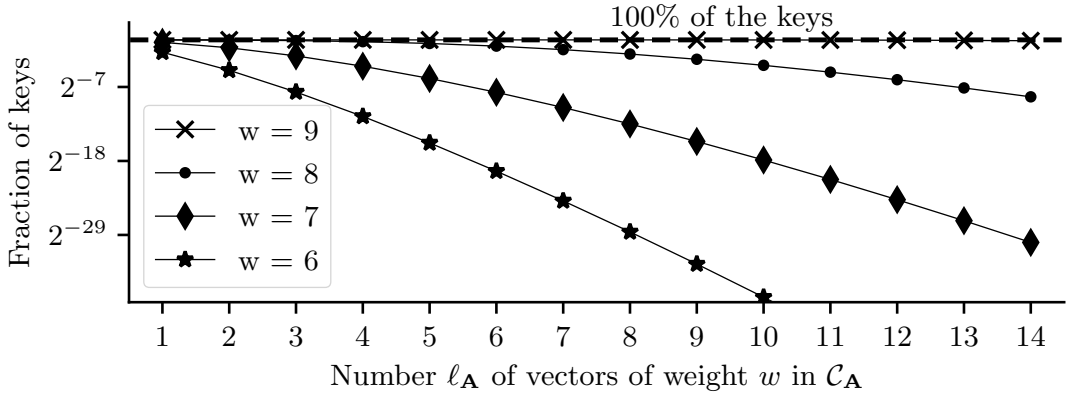

Figure 4.5: Fraction of the keys generated with BPKP–76 parameter set against which the attack is successful using different attack parameters $(w, \ell_{\mathbf{A}})$.

w	$\ell_{\mathbf{A}}$	$\hat{\alpha}$	Fraction of keys	Predicted work factor (matrix-vector products)	Empirical estimate (clock cycles)
5	14	1	0	$2^{39.46}$	$2^{34.39}$
6	11	2	$2^{-43.32}$	$2^{49.75}$	$2^{47.58}$
7	10	2	$2^{-17.86}$	$2^{55.84}$	$2^{48.62}$
8	9	3	$2^{-2.88}$	$2^{62.28}$	$2^{60.54}$
9	9	3	$2^{-0.00}$	$2^{64.16}$	$2^{62.31}$

Table 4.2: Estimates on the number of clock cycles necessary for a successful attack.

4.5.5 Experimental Results

To validate our proposed attack, we implemented it in SageMath and in C language, using M4RI [M. 12] library for efficient binary linear algebra computations. The source code is publicly available at www.ime.usp.br/~tpaiva.

Table 4.2 shows the performance of the attack against BPKP-76. To obtain empirical estimates on its performance, we considered the average number of clock cycles for the smallest level $\hat{\alpha}$ in the search for which we can get a significant number of samples. Then, the empirical estimate is given by the product between this average number of clock cycles and the average number of total nodes in level $\hat{\alpha}$. Thus smallest values of α give more accurate results. The values of $\ell_{\mathbf{A}}$ are chosen as to guarantee that the number of permutations to test is within reasonable computational limits and so that $\hat{\alpha} \leq 3$.

Notice how, in general, the estimates on the work factor of the attack tend to overestimate the real complexity of the attack. The main explanation for this fact seems to be that, for sufficiently large $\ell_{\mathbf{A}}$, the algorithm rarely enters in a leaf node, which is where most of the matrix product operations occur. This is exemplified by the decay, shown in Figure 4.2, of the curve representing the observed number of nodes in each level during a real attack, where, after level $\alpha = 7$, a node rarely has more than one child.

4.6 Asymptotic analysis

In the previous section, a detailed analysis of the attack is presented. However, the concrete analysis fails to provide a general idea of how the complexity grows, as the complexity of the components are not easy to simplify and must be computed using iterative procedures for products of binomial coefficients. Therefore we aim, in this section, to give simpler and closed expressions for the asymptotic attack complexity, but without compromising the reliability of the analysis.

4.6.1 Asymptotic growth of the attack parameters

First let us recall the growth of parameters m and ℓ with respect to n. To ensure that the binary PKP instances are difficult to solve on average, we need that, out of the $n!$ possible permutations of the rows of $\mathbf{V}$, about only one of them is in the kernel of $\mathbf{A}$. The dimension of $\mathbf{A}$ is m, which means the probability that a random vector belongs to

the kernel is $2^{(n-m)}/2^n = 2^{-m}$. Therefore, since the binary PKP is solved only when all ℓ column vectors of $\mathbf{V}_\pi$ are in the kernel of $\mathbf{A}$, then $n!2^{-m\ell} \approx 1$. As suggested by Lampe and Patarin [LP12], we consider that m and ℓ should be roughly the same size

$$m \approx \ell \approx \sqrt{\log n!} \leq \sqrt{n \log n}. \tag{4.5}$$

From Equation 4.5, we see that the dimension m of the code generated by $\mathbf{A}$ grows much slower than its size n. Intuitively then, as n gets larger, it gets harder to obtain codewords of weight much smaller than $n/2$, because of the small dimension. Since we need to deal with values of w close to $n/2$, we are interested in using the following lemma that gives approximations on binomial coefficients $\binom{n}{w}$ under this regime.

Lemma 4.6.1 (Eq. 5.41 [Spe14])**.** *Let n and w be positive integers such that $|n/2 - w| = o(n^{2/3})$. Then*

$$\binom{n}{w} \sim 2^n \sqrt{\frac{2}{n\pi}} e^{\frac{-(n-2w)^2}{2n}}.$$

We are now ready to show, in the following lemma, how to carefully choose values of w such that Lemma 4.6.1 ensures us that $\mathcal{L}_{\mathbf{A}}^w$ has a reasonable number of vectors.

Lemma 4.6.2. *Take the attack parameter w as*

$$w = \left\lfloor \frac{n}{2} - \sqrt{\frac{mn^{4/5}}{2 \log e}} \right\rceil. \tag{4.6}$$

Then, on average, the attack can effectively use parameters $(w, \ell_{\mathbf{A}})$ when $\ell_{\mathbf{A}}$ is smaller than

$$\ell_{\mathbf{A}} \leq \left(\sqrt{\frac{2}{\pi}} \right) 2^{\left(m\left(1-n^{(-1/5)}\right) - (\log n)/2\right)}.$$

Proof. Let $\mathbf{A}$ be an $m \times n$ random binary PKP public matrix. The attack parameters $(w, \ell_{\mathbf{A}})$ are effective when $\ell_{\mathbf{A}}$ is smaller than or equal to the number of vectors of weight w in the code generated by $\mathbf{A}$. Therefore, on average, the attack works when

$$\ell_{\mathbf{A}} \leq 2^{m-n} \binom{n}{w}.$$

Now take w as defined by Equation 4.6, and notice that, since $m \leq \sqrt{n \log n}$, from Equation 4.5, then

$$\begin{aligned} |n/2 - w| &= \sqrt{\frac{mn^{4/5}}{2 \log e}} \\ &\leq \sqrt{n^{4/5} \sqrt{n \log n}} \\ &= n^{13/20} (\log n)^{1/4} = o\left(n^{13/20+\epsilon}\right) \\ &= o\left(n^{2/3}\right). \end{aligned}$$

Therefore we can use Lemma 4.6.1 to obtain the approximation

$$2^{m-n}\binom{n}{w} \approx 2^{m-n}\left(2^n\sqrt{\frac{2}{n\pi}}e^{\frac{-(n-2w)^2}{2n}}\right) = 2^m\left(\sqrt{\frac{2}{n\pi}}e^{\frac{-(n-2w)^2}{2n}}\right).$$

But notice that

$$e^{\frac{-(n-2w)^2}{2n}} = e^{\frac{-(n/2-w)^2}{n/2}} = \exp\left(-\frac{1}{n/2}\sqrt{\frac{mn^{4/5}}{2\log e}}^2\right) = \exp\left(-\frac{mn^{-1/5}}{\log e}\right).$$

That is

$$e^{\frac{-(n-2w)^2}{2n}} = 2^{-mn^{-1/5}}. \tag{4.7}$$

Therefore the attack is effective for

$$\ell_{\mathbf{A}} \leq 2^m\left(\sqrt{\frac{2}{n\pi}}2^{-mn^{-1/5}}\right) = \left(\sqrt{\frac{2}{\pi}}\right)2^{\left(m\left(1-n^{(-1/5)}\right)-(\log n)/2\right)}.$$

□

Notice that when w is chosen according to the lemma above, then w/n approaches $1/2$ asymptotically when n gets larger, because

$$\begin{aligned}
\lim_{n\to\infty}\frac{w}{n} &= \lim_{n\to\infty}\left(\frac{1}{n}\right)\left(\frac{n}{2}-\sqrt{\frac{mn^{4/5}}{2\log e}}\right)\\
&= \frac{n}{2} - \lim_{n\to\infty}\left(\frac{1}{n}\right)\sqrt{\frac{mn^{4/5}}{2\log e}}\\
&= \frac{n}{2} - \lim_{n\to\infty}\sqrt{\frac{mn^{4/5}}{2n^2\log e}}\\
&= \frac{n}{2} - \lim_{n\to\infty}\sqrt{\frac{m}{2n^{1.2}\log e}}\\
&\geq \frac{n}{2} - \lim_{n\to\infty}\sqrt{\frac{\sqrt{n\log n}}{2n^{1.2}\log e}}\\
&= \frac{n}{2} - \lim_{n\to\infty}\sqrt{\frac{\sqrt{\log n}}{2n^{0.7}\log e}}\\
&= \frac{n}{2}, \text{ since } n^{0.7} \text{ grows much faster than } \sqrt{\log n}.
\end{aligned}$$

This motivates the following corollary, which has an important role in simplifying the analysis.

Corollary 4.6.2.1. *As n gets larger and w is taken as in Lemma 4.6.2, the values of $p(k,\alpha)$ stop depending on k, and we have*

$$p(k,\alpha) = \left(\frac{1}{2}\right)^k \left(1-\frac{1}{2}\right)^{\alpha-k} = 2^{-\alpha}.$$

□

As a first application of Corollary 4.6.2.1, we show that, for sufficiently large n, we do not need $\ell_{\mathbf{A}}$ to be very large. With roughly $\ell_{\mathbf{A}} \approx \log n$, the number $\mathbf{WF}_{\text{PERMS}}$ of permutations to test after each matching is very close to 1.

Lemma 4.6.3. *Consider binary PKP parameters (n,m,ℓ). Take attack parameters w as in Lemma 4.6.2 and $\ell_{\mathbf{A}} \geq \lceil \log n \rceil$. Then, for sufficiently large values of n, the average number of permutations to test after each matching is*

$$\mathbf{WF}_{\text{PERMS}} = 1.$$

Proof. From our concrete analysis, we know that

$$\mathbf{WF}_{\text{PERMS}} = \prod_{k=0}^{\ell_{\mathbf{A}}} \left(\frac{(np(k,\ell_{\mathbf{A}}))!}{(mp(k,\ell_{\mathbf{A}}))!} \right)^{\binom{\ell_{\mathbf{A}}}{k}}.$$

But Corollary 4.6.2.1 tells us that $p(k,\ell_{\mathbf{A}}) \approx 2^{-\ell_{\mathbf{A}}}$ when n is large. Therefore,

$$\mathbf{WF}_{\text{PERMS}} = \prod_{k=0}^{\ell_{\mathbf{A}}} \left(\frac{(n2^{-\ell_{\mathbf{A}}})!}{(m2^{-\ell_{\mathbf{A}}})!} \right)^{\binom{\ell_{\mathbf{A}}}{k}} = \prod_{k=0}^{\ell_{\mathbf{A}}} \left(\frac{(n2^{-\lceil \log n \rceil})!}{(m2^{-\lceil \log n \rceil})!} \right)^{\binom{\lceil \log n \rceil}{k}} = 1.$$

□

It is important to understand that the lemma above needs a relatively large n, because it uses Corollary 4.6.2.1. Therefore, to lower the number of permutations to test after each matching when attacking small values of n, we typically want to use $\ell_{\mathbf{A}}$ near the maximum provided by Lemma 4.6.2. Notice that even for relatively small values of n, there are usually more than $\log n$ vectors of weight w in the code generated by $\mathbf{A}$. For example, when $n = 38$ we have

$$\log(n) \approx 5.25 < 5.99 \approx \sqrt{\tfrac{2}{\pi}} 2^{m(1-n^{-1/5})-(\log n)/2}.$$

We are now ready to derive the asymptotic complexity of $\mathbf{WF}_{\text{SEARCH}}$, which is the most critical step of the attack.

4.6.2 Searching for matchings

Let us begin by deriving an asymptotic bound on the number of child nodes in each level of the search tree.

Lemma 4.6.4. *Take the attack parameter w as in Lemma 4.6.2. Then, for sufficiently large values of n, the number of child nodes in each level α of the search is given as*

$$\left|\hat{P}_{\alpha+1}\right| = \begin{cases} 2^{n-\ell-mn^{-1/5}} \left(2^{\alpha 2^{\alpha}/2}\right) \sqrt{\frac{2}{n\pi}}^{2^{\alpha}} & \text{if } \alpha \leq (\lceil \log n \rceil - 2); \\ 1 & \text{otherwise.} \end{cases}$$

Proof. By our concrete analysis, we know that $\left|\hat{P}_{\alpha+1}\right|$ is given as

$$\begin{aligned} \left|\hat{P}_{\alpha+1}\right| = \hat{\ell}_{\mathbf{K}} \overline{q}_{\alpha+1} &= \left(2^{-\ell}\binom{n}{k}\right) \frac{1}{\binom{n}{w}} \prod_{k=0}^{\alpha} \binom{np\,(k,\alpha)}{wp\,(k,\alpha)}^{\binom{\alpha}{k}} \\ &= 2^{-\ell} \prod_{k=0}^{\alpha} \binom{np\,(k,\alpha)}{wp\,(k,\alpha)}^{\binom{\alpha}{k}}. \end{aligned}$$

Using Corollary 4.6.2.1, we can simplify the above expression as

$$\begin{aligned} \left|\hat{P}_{\alpha+1}\right| &= 2^{-\ell} \prod_{k=0}^{\alpha} \binom{n2^{-\alpha}}{w2^{-\alpha}}^{\binom{\alpha}{k}} = 2^{-\ell} \binom{n2^{-\alpha}}{w2^{-\alpha}}^{\left(\sum_{k=0}^{\alpha}\binom{\alpha}{k}\right)} \\ &= 2^{-\ell} \binom{n2^{-\alpha}}{w2^{-\alpha}}^{2^{\alpha}}. \end{aligned}$$

Now, if $\alpha \geq (\lceil \log n \rceil - 1)$, then $w2^{-\alpha} < 1$, and

$$\binom{n2^{-\alpha}}{w2^{-\alpha}}^{2^{\alpha}} \leq \binom{n2^{-\alpha}}{\frac{n}{2}2^{-\alpha}}^{2^{\alpha}} \approx 1.$$

Therefore, we can focus on approximating the case when $\alpha \leq (\lceil \log n \rceil - 2)$. Remember that w is close to $n/2$, thus we can use Lemma 4.6.1 to get the approximation

$$\begin{aligned} \binom{n2^{-\alpha}}{w2^{-\alpha}} &\approx 2^{n2^{-\alpha}} \sqrt{\frac{2}{n2^{-\alpha}\pi}} e^{\frac{-(n2^{-\alpha}-2w2^{-\alpha})^2}{2n2^{-\alpha}}} \\ &= 2^{n2^{-\alpha}} 2^{\alpha/2} \sqrt{\frac{2}{n\pi}} e^{\frac{-(n-w)^2}{2n}2^{-\alpha}}. \end{aligned}$$

Recall Equation 4.7, which lets us further simplify the expression above as

$$\binom{n2^{-\alpha}}{w2^{-\alpha}} = 2^{n2^{-\alpha}} 2^{\alpha/2} \sqrt{\frac{2}{n\pi}} \left(2^{-mn^{-1/5}}\right)^{2^{-\alpha}}.$$

Now, getting back to $\left|\hat{P}_{\alpha+1}\right|$, we have

$$\begin{aligned}\left|\hat{P}_{\alpha+1}\right| &= 2^{-\ell}\binom{n2^{-\alpha}}{w2^{-\alpha}}^{2^\alpha} \\ &= 2^{-\ell}\left(2^{n2^{-\alpha}}2^{\alpha/2}\sqrt{\frac{2}{n\pi}}\left(2^{-mn^{-1/5}}\right)^{2^{-\alpha}}\right)^{2^\alpha} \\ &= 2^{n-\ell-mn^{-1/5}}\left(2^{\alpha 2^\alpha/2}\right)\sqrt{\frac{2}{n\pi}}^{2^\alpha}.\end{aligned}$$

□

Now that we have bounded the number of nodes in each level, we are ready to give the asymptotic bound on the search procedure.

Lemma 4.6.5. *Take attack parameters w and $\ell_{\mathbf{A}} \geq \lceil \log n \rceil$ as in Lemma 4.6.2. Then, the asymptotic work factor of the search is given as*

$$\mathbf{WF}_{\text{Search}} \approx 2^{\left(n-\ell-mn^{-1/5}\right)(\lceil \log n \rceil - 1) - 0.91n + \frac{1}{2}\log n + 1.33}.$$

Proof. From our concrete analysis, we know that the complexity of the search is the product of the number of nodes in each level of the search. Furthermore, Lemma 4.6.4 says that we only need to compute these values for $\alpha \leq \lceil \log n \rceil - 2$, because after this point, typically there is at most one possible child node. Therefore, the complexity of the search is given as

$$\begin{aligned}\mathbf{WF}_{\text{Search}} &= \prod_{\alpha=0}^{\ell_{\mathbf{A}}-1}\left|\hat{P}_{\alpha+1}\right| = \prod_{\alpha=0}^{\lceil \log n \rceil - 2}\left|\hat{P}_{\alpha+1}\right| \\ &= \prod_{\alpha=0}^{\lceil \log n \rceil - 2}\left(2^{n-\ell-mn^{-1/5}}\left(2^{\alpha 2^\alpha/2}\right)\sqrt{\frac{2}{n\pi}}^{2^\alpha}\right) \\ &= 2^{\left(n-\ell-mn^{-1/5}\right)(\lceil \log n \rceil - 1)}2^{\left(\sum_{\alpha=0}^{\lceil \log n \rceil - 2}\alpha 2^\alpha/2\right)}\left(\frac{2}{n\pi}\right)^{\left(\sum_{\alpha=0}^{\lceil \log n \rceil - 2}2^\alpha/2\right)} \\ &= 2^{\left(n-\ell-mn^{-1/5}\right)(\lceil \log n \rceil - 1)}2^{\left(1+\frac{n}{4}(\lceil \log n \rceil - 3)\right)}2^{\left(\log\frac{2}{n\pi}\right)(n/4-1/2)} \\ &\approx 2^{\left(n-\ell-mn^{-1/5}\right)(\lceil \log n \rceil - 1) - 0.91n + \frac{1}{2}\log n + 1.33}.\end{aligned}$$

□

4.6.3 Asymptotic complexity of the attack

We are almost ready to complete the asymptotic analysis of the attack. The only missing component to consider is $\mathbf{WF}_{\text{LowWeightSets}}$. Using the bruteforce algorithm, one needs to test, for all $\binom{n}{w} = O(2^n)$ possible vectors of weight w, if they are in the space generated

by $\mathbf{A}$ or in the left kernel of $\mathbf{V}$. Therefore, the complexity of building sets $\mathcal{L}^w_{\mathbf{A}}$ and $\mathcal{L}^w_{\mathbf{K}}$ is

$$\mathbf{WF}_{\text{LowWeightSets}} = O(2^n).$$

We can then combine the result above with Lemmas 4.6.3 and 4.6.5 to obtain the complexity of the attack, as given next.

Lemma 4.6.6. *Take attack parameters w and $\ell_{\mathbf{A}} \geq \lceil \log n \rceil$ as in Lemma 4.6.2. Then, the asymptotic work factor of the attack is given as*

$$\begin{aligned}\mathbf{WF}_{\text{Attack}} &= \mathbf{WF}_{\text{LowWeightSets}} + (\mathbf{WF}_{\text{Search}})\,(\mathbf{WF}_{\text{Perms}}) \\ &= O\left(2^{\left(n-\ell-mn^{-1/5}\right)(\lceil \log n \rceil - 1) - 0.91n + \frac{1}{2}\log n}\right).\end{aligned}$$

□

Figure 4.6 shows how the asymptotic complexity presented above compares with the simulations based on the concrete analysis. We can see that the asymptotic estimate appears to be realistic, even though the ceiling operation used for $\lceil \log n \rceil$ makes the function rapidly increase when $n-1$ is a power of 2, and then decrease until the next power of 2 is found.

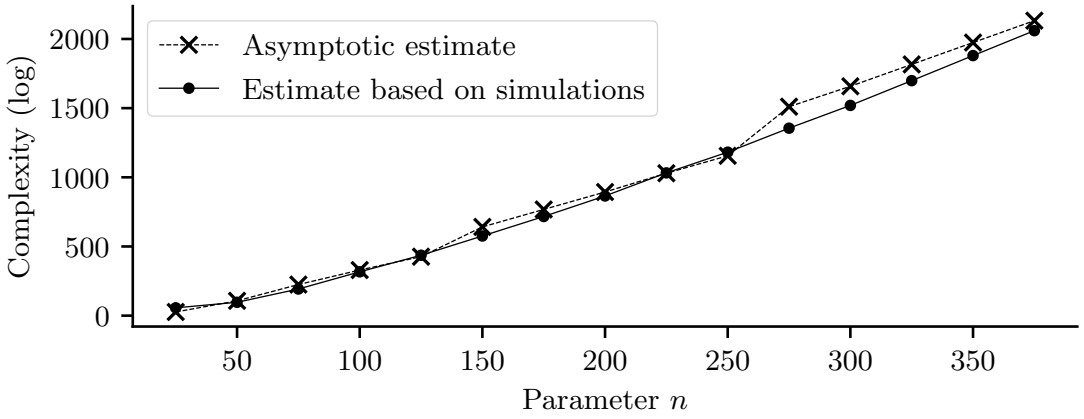

Figure 4.6: Asymptotic complexity of the attack.

Figure 4.7 shows an asymptotic comparison between our algorithm and the one by Koussa et al. [KMRP19]. Even though their algorithm is currently the best generic algorithm for solving PKP in every field, we can see that our algorithm has a considerable advantage in the binary case. To help us visualize the asymptotic growth of our attack, we consider a smooth version of the estimate that consists in using $\log n$ instead of $\lceil \log n \rceil$ in the expression provided in Lemma 4.6.6.

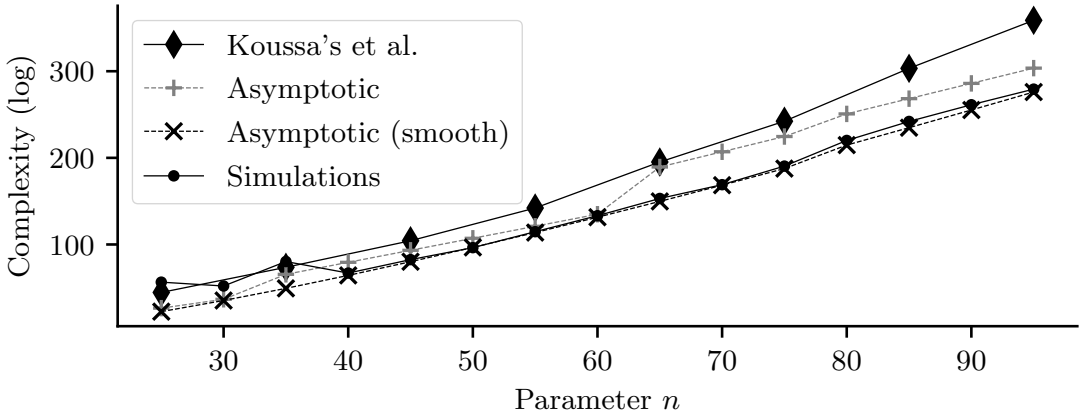

Figure 4.7: Comparison between our attack and the one by Koussa et al. [KMRP19].

4.7 On secure parameters for binary PKP

A conservative approach to select parameters for binary PKP, considering security level λ, would be to choose them in such a way that no class of keys that occurs with probability greater than $2^{-\lambda}$ should be attacked with less 2^{λ} operations. Furthermore, the choice of parameters should consider the use of binary PKP when building a signature scheme, and, as such, should aim to minimize not only the key sizes, but also signature sizes and the computational cost to sign and verify each signature.

The safest possible choice of parameters would be the ones that make it difficult to even build sets $\mathcal{L}^w_{\mathbf{A}}$ and $\mathcal{L}^w_{\mathbf{K}}$. If we take schemes that rely on the difficulty of finding small weight codewords such as MDPC [MTSB13], this would result in a very large matrix $\mathbf{A}$. This, however, would have a very negative impact on performance, key sizes and signature length.

A less conservative approach is to scale parameters (n, m, ℓ) and compute $\mathbf{WF}^{(n,m,\ell,w,\ell_{\mathbf{A}})}_{\text{Attack}}$ and $\mathbf{KF}^{(n,m,\ell,w,\ell_{\mathbf{A}})}_{\text{Attack}}$ for different attack parameters $(w, \ell_{\mathbf{A}})$. The search is efficient and can be done with the code that we provide. However, it is important to notice that it seems to be early to state sets of parameters for BPKP, as there may be some opportunities to improve this attack, which could thwart the security of parameters suggested without careful consideration. Our recommendation therefore is to avoid the Binary PKP, and more generally, the PKP using small fields for matrix $\mathbf{A}$, for which the search for low weight codewords can be done efficiently.

4.8 Conclusion and future work

In this paper, we present the first attack that targets binary PKP and provide a detailed analysis on the attack's components. The attack is practical and we provide an implementation of the attack in SageMath and C. Furthermore, the attack shows an inherently weakness of PKP using small fields, and we recommend that binary PKP be avoided while

its security is not well understood against this new type of attack.

For future work, we plan to extend this attack to the original PKP, hoping to better understand what is the minimum finite field size p that can used securely. Furthermore, we believe that there are some opportunities to improve this attack. For example, it may be possible to increase the fraction of keys that one can attack by considering different parameters w simultaneously, or one can try to reduce the complexity of matching by introducing heuristics.

Acknowledgments

The authors would like to thank Augusto C. Ferrari for his helpful comments on earlier drafts of this paper. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. This research is part of the INCT of the Future Internet for Smart Cities funded by CNPq proc. 465446/2014-0, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, FAPESP proc. 14/50937-1, and FAPESP proc. 15/24485-9.

Chapter 5

Faster constant-time decoder for MDPC codes and applications to BIKE KEM

Abstract. BIKE is a code-based key encapsulation mechanism (KEM) that was recently selected as an alternate candidate by the NIST's standardization process on post-quantum cryptography. This KEM is based on the Niederreiter scheme instantiated with QC-MDPC codes, and it uses the BGF decoder for key decapsulation. We discovered important limitations of BGF that we describe in detail, and then we propose a new decoding algorithm for QC-MDPC codes called PickyFix. Our decoder uses two auxiliary iterations that are significantly different from previous approaches and we show how they can be implemented efficiently. We analyze our decoder with respect to both its error correction capacity and its performance in practice. When compared to BGF, our constant-time implementation of PickyFix achieves speedups of 1.18, 1.29, and 1.47 for the security levels 128, 192 and 256, respectively.
Keywords: Post-quantum cryptography, BIKE, MDPC, LDPC, constant-time decoding

5.1 Introduction

BIKE [ABB+21] is a code-based key encapsulation mechanism (KEM) selected as an alternate candidate for the NIST post quantum standardization process. The scheme consists of a variant of the Niederreiter [Nie86] scheme using quasi-cyclic moderate-density parity-check (QC-MDPC) codes instead of Goppa codes. As such, BIKE can be seen as a refinement of Misoczki's et al. QC-MDPC McEliece [MTSB13].

The use of QC-MDCP [MTSB13] codes yields two advantages. The first one is that the public key is much smaller, since one needs only one row to represent a quasi-cyclic matrix in systematic form. The second is that matrix multiplication, and thus encoding, is much faster for quasi-cyclic matrices. However, QC-MDPC codes comes with an important disadvantage: their decoding algorithms have a non-zero probability of failure. This fact

was exploited in the famous GJS [GJS16] key-recovery reaction attack, that provided the ground for side-channel attacks against QC-MDPC [RHHM17] and further attacks against other code-based encryption schemes [SSPB19, FHS+17].

To deal with this problem, BIKE's original proposal [ABB+17b] used ephemeral keys. However, recent approaches on obtaining negligible decryption failure rate (DFR) [Til18, SV20a, Vas21], together with Hofheinz et al. [HHK17b] CCA security conversions that accounts for decryption errors, motivated BIKE proponents to consider key-reuse. In particular, Sendrier and Vasseur [SV20a, Vas21] propose a framework that, under reasonable assumptions, allows them to find parameters where the DFR should be negligible using experiments and statistical analysis. This framework was used in BIKE's last revision [ABB+21], which uses the state-of-the-art BGF decoder [DGK20c, DGK19] with parameters that supposedly achieve negligible DFR.

While trying to improve BGF's performance, we noticed two limitations. The first one is that its performance cannot be improved by considering a lower number of iterations, otherwise it breaks the main hypothesis for using Vasseur's extrapolation framework [Vas21]. The second is that some of its iterations can be made more efficient by merging them into one iteration. After analyzing BGF's strengths and weaknesses, we were able to derive a new and more efficient decoder.

Contribution. We propose a new decoding algorithm for QC-MDPC codes, called PickyFix. This decoder uses two auxiliary iterations that are significantly different from previous approaches: the FixFlip iteration, which flips a fixed number of bits, and the PickyFlip iteration, which uses different thresholds to flip ones and zeros. These iterations allow PickyFix to work with a lower number of iterations than BGF, which, together with our constant-time implementation, makes PickyFix achieve speedups of 1.18, 1.29, and 1.47 for the security levels 128, 192 and 256, respectively. The code and data are publicly available at https://github.com/thalespaiva/pickyfix.

Organization. We begin by quickly reminding some basic concepts from coding theory and QC-MDPC decoding in Section 5.2. BIKE and its security parameters are presented in Section 5.3. Then, in Section 5.4, we analyze BGF in detail to show its strengths and weaknesses. Our proposed decoder PickyFix is introduced in Section 5.5, and then we analyze its parameters and decoding performance in Section 5.6. In Section 5.7, we discuss how to implement PickyFix efficiently and in constant-time and then compare its performance with BGF. Finally, we conclude and discuss interesting future work in Section 5.8.

5.2 Background

Please notice that, although some of the concepts defined below already appeared in Section 2.5, the definitions are kept here to allow for readers to independently read this technical chapter.

A *binary* $[n, k]$*-linear code* is a k-dimensional linear subspace of $\mathbb{F}_2^n$, where $\mathbb{F}_2$ denotes the binary field. If $\mathcal{C}$ is a binary $[n, k]$-linear code spanned by the rows of a matrix $\mathbf{G}$ of $\mathbb{F}_2^{k\times n}$, we say that $\mathbf{G}$ is a *generator matrix* of $\mathcal{C}$. Similarly, if $\mathcal{C}$ is the kernel of a matrix $\mathbf{H}$ of $\mathbb{F}_2^{r\times n}$, we say that $\mathbf{H}$ is a *parity-check matrix* of $\mathcal{C}$. The Hamming *weight* of a vector $\mathbf{v}$, denoted by $\mathrm{w}(\mathbf{v})$, is the number of its non-zero entries. The *syndrome* $\mathbf{z}$ of a vector $\mathbf{e}$ with respect to a parity check matrix $\mathbf{H}$ is the vector $\mathbf{z} = \mathbf{e}\mathbf{H}^\top$. If the vector $\mathbf{e}$ is sufficiently sparse and the linear code defined by $\mathbf{H}$ is sufficiently good, it may be possible to recover $\mathbf{e}$ from the syndrome $\mathbf{z}$ by using efficient decoding algorithms. The support of a binary vector $\mathbf{v}$, denoted as $\mathrm{supp}(\mathbf{v})$, is the set $\mathrm{supp}(\mathbf{v}) = \{i : \mathbf{v}_i = 1\}$.

A moderate-density parity-check (MDPC) code [MTSB13] is a linear code that admits a moderately sparse parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{r\times n}$. The weight of each column of $\mathbf{H}$ is set to be all equal to a fixed value d, and require that $d = O(\sqrt{n})$. For applications in cryptography, it is particularly useful to consider quasi-cyclic MDPC (QC-MDPC) codes, because they allow for smaller keys and more efficient operations. BIKE [ABB+21] is defined over QC-MDPC codes with two circulant blocks, which are MDPC codes that admit a sparse parity check matrix of the form $\mathbf{H} = [\mathbf{H}_0|\mathbf{H}_1]$, where each $r \times r$ binary matrix $\mathbf{H}_0$ and $\mathbf{H}_1$ is circulant.

MDPC codes admit very efficient decoders, which are called bit-flipping decoders [Gal62]. All variants of bit-flipping decoders work based on the following observations. Let $\mathbf{e}$ be a sparse vector whose syndrome with respect to the sparse matrix $\mathbf{H}$ is $\mathbf{z} = \mathbf{e}\mathbf{H}^\top$. Suppose we do not know $\mathbf{e}$ but want to recover it from $\mathbf{z}$ using our knowledge from $\mathbf{H}$. We know that $\mathbf{z} = \sum_{i\in\mathrm{supp}(\mathbf{e})} \mathbf{H}_i^\top$, where $\mathbf{H}_i^\top$ denotes the transpose of the i-th column of $\mathbf{H}$.

Now, since $\mathbf{e}$ and each column $\mathbf{H}_i^\top$ are sparse, we can estimate the likelihood that $\mathbf{e}_i = 1$ by checking how closely $\mathbf{z}$ matches with column i of $\mathbf{H}$: the more they are similar, the higher is the probability that $\mathbf{e}_i = 1$. The similarity measure for each column i is what is known as the unsatisfied parity-check (UPC) counter, denoted as $\mathtt{upc}_i$, and it is equal to the size of the intersection of $\mathrm{supp}(\mathbf{z})$ and $\mathrm{supp}\left(\mathbf{H}_i^\top\right)$. The name UPC comes from the fact that the set $\mathrm{supp}(\mathbf{z})$ is sometimes called the set of unsatisfied equations, and therefore $\mathtt{upc}_i$ counts the number of unsatisfied equations that are caught by $\mathbf{H}_i^\top$.

Algorithm 5.1 shows the steps that a general bit-flipping algorithm performs when trying to obtain $\mathbf{e}$ from $\mathbf{z}$ and $\mathbf{H}$. The algorithm stops when it finds a vector $\hat{\mathbf{e}}$ with the same syndrome as $\mathbf{e}$, or if the number of iterations exceeds some limit. Notice that the partial syndrome defines the objective syndrome in each iteration, and in the ideal case, vector $\hat{\mathbf{e}}$ gets closer and closer to $\mathbf{e}$ after each iteration. Although most bit-flipping algorithms used in cryptography [Gal62, MTSB13, DGK19, SV19] can be framed in the general description above, they can vary significantly with respect to how the threshold for flipping bits is selected in each iteration.

Algorithm 5.1 General bit-flipping decoding algorithm.

procedure GENERALBITFLIPPING($\mathbf{z}, \mathbf{H}$)
  Start with the partial error vector $\hat{\mathbf{e}} \leftarrow \mathbf{0} \in \mathbb{F}_2^n$
  Initialize the number of iterations $\mathtt{it} \leftarrow 0$
  Let the partial syndrome $\mathbf{s} \leftarrow \mathbf{z} + \hat{\mathbf{e}}\mathbf{H} = \mathbf{z}$
  while $\mathbf{s} \neq \mathbf{0}$ and $\mathtt{it} <$ maximum number of iterations **do**
    Compute the UPC counters $\mathtt{upc}_i$ with respect to $\mathbf{s}$ and $\mathbf{H}$, for $i = 1$ to n
    For each $i = 1$ to n, flip bit $\hat{\mathbf{e}}_i$ if $\mathtt{upc}_i$ is above a certain threshold
    Update the partial syndrome $\mathbf{s} \leftarrow \mathbf{z} + \hat{\mathbf{e}}\mathbf{H}$
    $\mathtt{it} \leftarrow \mathtt{it} + 1$
  if $\mathbf{s} = \mathbf{0}$ **then**
    return $\hat{\mathbf{e}}$
  else
    return $\perp$, indicating that the maximum number of iterations was reached

5.3 BIKE

The purpose of a key encapsulation mechanism is to use public-key encryption algorithms to securely exchange a key between two parties. These parties can then use secret-key algorithms, which are much more efficient, to exchange large messages.

For a clearer presentation, we describe BIKE algorithms without the implicit-rejection Fujisaki-Okamoto transformation [HHK17b], usually denoted by FO$^{\not\perp}$. However, notice that when discussing the experimental performance of our algorithm in Section 5.7.3, we consider the full decapsulation with the FO$^{\not\perp}$ transformation applied.

5.3.1 Parameters and algorithms

Setup. On input 1^λ, where λ is the security level, the setup algorithm returns parameters r, w and t taken from the parameter Table 5.1. Parameters r and w will define the family of QC-MDPC codes to be used while t controls the weight of the error used for encryption, as will be detailed in the following sections. The table also provides the estimated decryption failure rates (DFR) for each parameters set according to Vasseur's framework [Vas21].

Parameter set	Security level λ	r	w	$d = w/2$	t	Decoder	DFR estimate
BIKE Level 1	128	12323	142	71	134	BGF	2^{-128}
BIKE Level 3	192	24659	206	103	199	BGF	2^{-192}
BIKE Level 5	256	40973	274	137	264	BGF	2^{-256}

Table 5.1: BIKE parameters for each security level.

Key Generation. Let $\mathbf{h}_0$ and $\mathbf{h}_1$ be two vectors of r bits of odd weight $d = w/2$. Build the circulant matrices $\mathbf{H}_0$ and $\mathbf{H}_1$ by taking $\mathbf{h}_0$ and $\mathbf{h}_1$ as their first rows, correspondingly. If $\mathbf{H}_1$ is not invertible, restart the process by selecting another $\mathbf{h}_1$. Then the secret key is

the sparse matrix $\mathbf{H} = [\,\mathbf{H}_0 \,|\, \mathbf{H}_1\,] \in \mathbb{F}_2^{r\times 2r}$ and the public key is the dense circulant matrix $\mathbf{H}_{\text{Pub}} = \mathbf{H}_1\mathbf{H}_0^{-1}$.

Notice that matrices $\mathbf{H}$ and $[\,\mathbf{I} \,|\, \mathbf{H}_{\text{Pub}}\,]$ are both parity checks of the same quasi-cyclic linear code. However, the sparsity of the first one allows for efficient syndrome decoding using bit-flipping algorithms.

Encapsulation. Select two random binary vectors $\mathbf{e}_0$ and $\mathbf{e}_1$ such that $\mathrm{w}(\mathbf{e}_0) + \mathrm{w}(\mathbf{e}_1) = t$. Then the key to be shared is $k_{\text{Shared}} = \mathcal{H}([\mathbf{e}_0|\mathbf{e}_1])$, for some cryptographic hash function $\mathcal{H}$. To encapsulate the key k_{Shared}, compute the ciphertext $\mathbf{c} = \mathbf{e}_0 + \mathbf{e}_1\mathbf{H}_{\text{Pub}}^\top \in \mathbb{F}_2^r$. Notice that ciphertext $\mathbf{c}$ then corresponds to the syndrome of the low weight vector $[\mathbf{e}_0|\mathbf{e}_1]$ with respect to the public parity-check matrix $[\,\mathbf{I} \,|\, \mathbf{H}_{\text{Pub}}\,]$.

Decapsulation. Given the ciphertext $\mathbf{c}$, the receiver, who knows the sparse parity-check matrix $\mathbf{H}$, first compute the secret syndrome $\mathbf{z} = \mathbf{c}\mathbf{H}_0^\top$. Notice that

$$\mathbf{z} = \mathbf{c}\mathbf{H}_0^\top = \left(\mathbf{e}_0 + \mathbf{e}_1\mathbf{H}_{\text{Pub}}^\top\right)\mathbf{H}_0^\top = \mathbf{e}_0\mathbf{H}_0^\top + \mathbf{e}_1\mathbf{H}_{\text{Pub}}^\top\mathbf{H}_0^\top = \mathbf{e}_0\mathbf{H}_0^\top + \mathbf{e}_1\mathbf{H}_1^\top.$$

Therefore, as mentioned by the end of Section 5.2, the receiver can use some QC-MDPC bit-flipping decoding algorithm, together with their knowledge of the secret matrix $\mathbf{H}$ to recover the sparse vector $[\mathbf{e}_0|\mathbf{e}_1]$ and compute the shared key $k_{\text{Shared}} = \mathcal{H}([\mathbf{e}_0|\mathbf{e}_1])$.

In the last revision of BIKE [ABB+21], the authors recommend the BGF decoding algorithm [DGK20c], which is the state-of-the art QC-MDPC decoder. Before introducing BGF, let us first discuss the security of BIKE and, in particular, why good decoders are very important to ensure BIKE's security.

5.3.2 Security and negligible decryption failure rate

The security of the scheme is based on three hypotheses. The first two are standard conjectures for quasi-cyclic codes, namely the hardness of the syndrome decoding problem and the hardness of finding codewords of a fixed low weight. This ensures that one can neither recover the secret sparse matrices $\mathbf{H}_0$ and $\mathbf{H}_1$ from $\mathbf{H}$, nor the secret message $[\mathbf{e}_0|\mathbf{e}_1]$ from ciphertext $\mathbf{c}$. The third hypothesis is that the decryption failure rate (DFR) is negligible with respect to the security parameter. Although we cannot prove the third hypothesis, Vasseur [Vas21] proposed a framework that, under weaker hypothesis, allows one to get confident that some decoders achieve negligible DFR for selected parameter sets.

It is shown [TS16, Sen11] that parameters t and w are the most important when determining the security level, since they control the weight of the sparse vectors. Intuitively, if w or t are too small, it is easy to find $\mathbf{h}_0$ or the partial encryption error $\mathbf{e}_0$ by enumerating low weight vectors. But they may not be so large, with respect to r, otherwise the probability of failing to decrypt a ciphertext would be too high. Therefore, to define parameters (t, w, r), one typically fixes (t, w) sufficiently large to achieve high security levels, and then define r such that the decryption failure rate is low enough for the desired application.

In 2016, Guo et al. [GJS16] showed that decryption failures could lead to a full key recovery attack against schemes based on QC-MDPC codes. To deal with the potential vulnerability faced by schemes within which decryption failures occur, Hofheinz et al. [HHK17b] refined the Fujisaki-Okamoto [FO99] transformation showing that a scheme whose decryption failure rate is below $2^{-\lambda}$ can be transformed into a CCA secure one.

Unlike for algebraic codes, such as Goppa or Reed-Solomon codes, whose decoders are guaranteed to decode all errors in vectors up to a given weight, we cannot yet give strong mathematical guarantees on the error correction capability of decoders for QC-MDPC codes. Recently, Sendrier and Vasseur [SV20a, Vas21] proposed a method that, under reasonable hypotheses, allows one to use simulations and simple statistical analysis to find parameters (r, t, w) such that a QC-MDPC decoder fails with negligible probability with respect to some security parameters λ.

Let t and w be fixed positive integers and let us consider a hypothetical QC-MDPC decoder $\mathcal{D}$. Let $\mathrm{DFR}_{\mathcal{D}}(r)$ denote the decryption failure rate of $\mathcal{D}$ when decrypting a ciphertext generated at random with respect to a random QC-MDPC key with parameters (r, t, w). The main observation by Sendrier and Vasseur [SV20a] is that the curve $\log_2(\mathrm{DFR}_{\mathcal{D}}(r))$ is typically concave for practical QC-MDPC decoders and for all values of r such that $\mathrm{DFR}_{\mathcal{D}}(r)$ is high enough so that failures can be observed in simulations. Vasseur's [Vas21] model then makes the following assumption: for a given decoder $\mathcal{D}$ and security level λ, the curve $\log_2(\mathrm{DFR}_{\mathcal{D}}(r))$ is concave in the region where $\mathrm{DFR}_{\mathcal{D}}(r) \geq 2^{-\lambda}$. This assumption is somewhat consistent with Tillich's [Til18] asymptotic theoretical model for MDPC codes, which shows that the dominating term in $\log_2(\mathrm{DFR}_{\mathcal{D}}(r))$ decreases linearly with r.

Figure 5.1 illustrates how Vasseur's [Vas21] model can be used to estimate the block parameter r that allows for negligible failure rate with respect to the security parameter $\lambda = 128$. First, one performs DFR simulations for increasing values of r until it cannot see any decoding failure. Then, they take the last two points (r_A, p_A) and (r_B, p_B) in the $\log_2$ DFR plot such that a number of failures were observed and compute the line passing through them. According to the extrapolation hypothesis, the decoder fails with negligible probability for $r = r_{\mathrm{ext}}$, the point where the line intercepts $\mathrm{DFR} = 2^{-\lambda}$. Finally, choose parameter r to be the least prime $r \geq r_{\mathrm{ext}}$ such that 2 is primitive modulo r. This avoids both squaring attacks [LJS+16] and other potential attacks based on the factorization of the cyclic polynomial ring[25] $\mathbb{F}_2[X]/(X^r - 1)$.

Since there is always some error in the DFR estimates, Vasseur [Vas21] uses confidence intervals for the observed DFR and compute a conservative extrapolation for r as follows. Let p_A and p_B be the DFRs for r_A and r_B, respectively, where $r_A < r_B$. Consider p_A^- and p_B^+ to be the lower and upper limit for p_A and p_B according to Binomial confidence intervals for p_A and p_B. Then a conservative extrapolation for r_{ext} is obtained by considering the line passing through (r_A, p_A^-) and (r_B, p_B^+). Vasseur [Vas21] uses the Clopper-Pearson confidence interval together with posterior probabilities to obtain a narrower interval, with confidence

[25] The cyclic polynomial ring $\mathbb{F}_2[X]/(X^r - 1)$ is isomorphic to the ring of circulant matrices used in BIKE.

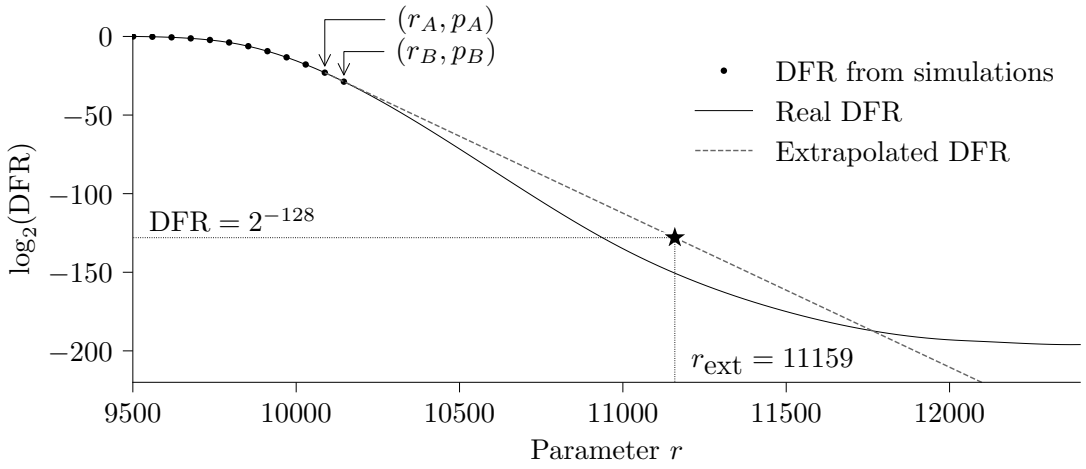

Figure 5.1: Illustration of Vasseur's [Vas21] DFR extrapolation framework considering 128 bits of security and a hypothetical decoder.

level $\alpha = 0.01$. In this work we use the same α with the Clopper-Pearson interval, but we do not use the posterior probabilities. Even though this tends to give slightly more conservative estimates, it is easier to compute.

5.3.3 BGF: State-of-the-art QC-MDPC decoder

BGF [DGK20c], which stands for Black-Gray-Flip, is one of the most efficient known decoders for QC-MDPC codes. This decoder is an improvement of the Black-Gray decoder first proposed by Sendrier and Misoczki in a previous version[26] of CAKE [BGG+17], a predecessor of BIKE.

As a decoding algorithm, BGF's goal is to, given a syndrome ciphertext $\mathbf{c} = \mathbf{e}_0 + \mathbf{e}_1\mathbf{H}_{\text{Pub}}^\top$, recover the sparse error vector $\mathbf{e} = [\mathbf{e}_0|\mathbf{e}_1]$ using the secret sparse matrix $\mathbf{H}$. The algorithm first computes the secret syndrome $\mathbf{z} = \mathbf{c}\mathbf{H}_0^\top$, then starts with $\mathbf{e} \leftarrow \mathbf{0}$ and performs a sequence of N_{Iter} iterations, each of which updates its knowledge on $\mathbf{e}$ until either $\mathbf{z} = \mathbf{e}\mathbf{H}^\top$ or the number of iterations exceeds a certain limit N_{Iter} and a decoding failure occurs. Before introducing BGF, let us first define its auxiliary procedures.

BGF Auxiliary Algorithms. BGF uses two bit-flipping auxiliary procedures: BitFlipIter and BitFlipMaskedIter, which are formally described in Algorithm 5.2. These procedures are very similar to other iterative decoders, such as the original Gallager's bit-flipping algorithm [Gal62].

Both algorithms flip bits of the partial error vector $\mathbf{e}$ when their corresponding UPC counters are above some threshold, τ_0 for BitFlipIter and τ_1 for BitFlipMaskedIter. However they differ in some important points. First, BitFlipIter not only flips the bits, but it also marks the bits in either black or gray, using bit-masks `BlackMask` and `GrayMask`.

[26] Unfortunately, there appears to be no reference to the version in which the Black-Gray decoder appeared.

Black bits are the ones that are flipped with a somewhat high confidence ($\texttt{upc}_j \geq \tau_0$), while gray bits are the ones that were almost selected for flipping ($\tau_0 > \texttt{upc}_j \geq \tau_0 - \delta$), but did not make it because of a minor difference δ. On the other hand, BitFlipMaskedIter is a simple bit flip iteration based on the UPC value, but it only flips bits that are marked 1 in a given mask `Mask`.

Algorithm 5.2 Auxiliary iterations used by BGF.

procedure BitFlipIter($\mathbf{H}, \mathbf{z}, \mathbf{e}, \mathbf{s}, \tau_0$)
  `BlackMask` $\leftarrow \mathbf{0} \in \mathbb{F}_2^{2r}$
  `GrayMask` $\leftarrow \mathbf{0} \in \mathbb{F}_2^{2r}$
  for $j = 1$ to $2r$ **do**
    $\texttt{upc}_j \leftarrow \left|\text{supp}\left(\mathbf{s}\right) \cap \text{supp}\left(\mathbf{H}_j^\top\right)\right|$
    if $\texttt{upc}_j \geq \tau_0$ **then**
      $\mathbf{e}_j \leftarrow \overline{\mathbf{e}_j}$ ▷ Flips coordinate j of $\mathbf{e}$
      $\texttt{BlackMask}_j \leftarrow 1$
    else if $\texttt{upc}_j \geq \tau_0 - \delta$ **then**
      $\texttt{GrayMask}_j \leftarrow 1$
  $\mathbf{s} \leftarrow \mathbf{z} + \mathbf{e}\mathbf{H}^\top$ ▷ Recomputes the partial syndrome
  return $\mathbf{e}$, $\mathbf{s}$, `BlackMask`, `GrayMask`

procedure BitFlipMaskedIter($\mathbf{H}, \mathbf{z}, \mathbf{e}, \mathbf{s}, \texttt{Mask}, \tau_1$)
  for $j = 1$ to $2r$ **do**
    $\texttt{upc}_j \leftarrow \left|\text{supp}\left(\mathbf{s}\right) \cap \text{supp}\left(\mathbf{H}_j^\top\right)\right|$
    if $\texttt{Mask}_j = 1$ and $\texttt{upc}_j \geq \tau_1$ **then**
      $\mathbf{e}_j \leftarrow \overline{\mathbf{e}_j}$ ▷ Flips coordinate j of $\mathbf{e}$
  $\mathbf{s} \leftarrow \mathbf{z} + \mathbf{e}\mathbf{H}^\top$ ▷ Recomputes the partial syndrome
  return $\mathbf{e}$, $\mathbf{s}$

The BGF algorithm. BGF is defined as Algorithm 5.3. Intuitively, the first call to BitFlipIter flips the bits for which it has a high confidence that they are wrong, by using a selective threshold function Thresh. Then it comes the two regret steps: first the black and then the gray. In the black regret, all the 1 bits added in the previous step that have an UPC strictly greater[27] than $(d+1)/2$ will be flipped back to 0. The gray regret step is analogous, but now over the bits marked in `GrayMask`, which are called gray bits. These consist of 0 bits that were not flipped in the first step because their UPC were smaller than, but somewhat close to, the selected threshold.

After the first and most costly iteration ensured a good start, hopefully with only a small number of errors left to be corrected, BGF continues with $N_{\text{Iter}} - 1$ iterations of BitFlipIter that will try to correct the remaining errors. Notice that the masks are not needed after this point, and thus, are ignored.

[27] Notice that this is done by choosing $\tau_1 = (d+1)/2 + 1$, since the flipping condition is $\texttt{upc}_j \geq \tau_1$.

Algorithm 5.3 The BGF decoding algorithm.

procedure BGF($\mathbf{H} = [\mathbf{H}_0|\mathbf{H}_1], \mathbf{z} = \mathbf{c}\mathbf{H}_0^\top$)
  $\mathbf{e} \leftarrow \mathbf{0} \in \mathbb{F}_2^{2r}$ ▷ Initializes the partial error vector
  $\mathbf{s} \leftarrow \mathbf{z}$ ▷ Initializes the partial syndrome
  for $i = 1$ to N_{Iter} **do**
    ▷ Every time $\mathbf{e}$ and $\mathbf{s}$ are updated, it holds that $\mathbf{s} = \mathbf{z} + \mathbf{e}\mathbf{H}^\top$
    if $i = 1$ **then**
      $\mathbf{e}, \mathbf{s}, \texttt{BlackMask}, \texttt{GrayMask} \leftarrow \text{BitFlipIter}(\mathbf{H}, \mathbf{z}, \mathbf{e}, \mathbf{s}, \tau_0 = \text{Thresh}(\mathbf{s}))$
      $\mathbf{e}, \mathbf{s} \leftarrow \text{BitFlipMaskedIter}(\mathbf{H}, \mathbf{z}, \mathbf{e}, \mathbf{s}, \texttt{BlackMask}, \tau_1 = (d+1)/2+1)$
      $\mathbf{e}, \mathbf{s} \leftarrow \text{BitFlipMaskedIter}(\mathbf{H}, \mathbf{z}, \mathbf{e}, \mathbf{s}, \texttt{GrayMask}, \tau_1 = (d+1)/2+1)$
    else
      $\mathbf{e}, \mathbf{s} \leftarrow \text{BitFlipIter}(\mathbf{H}, \mathbf{z}, \mathbf{e}, \mathbf{s}, \tau_0 = \text{Thresh}(\mathbf{s}))$ ▷ Ignores the black-gray masks
  if $\mathbf{e}\mathbf{H}^\top = \mathbf{z}$ **then** ▷ This condition is equivalent to $\mathbf{s} = \mathbf{0}$
    return $\mathbf{e}$
  else
    return $\perp$ ▷ Decoding failure

BGF Parameters. Table 5.2 shows the parameters δ, N_{Iter} and threshold function Thresh proposed for the different security levels together with their performance under our platform[28]. We considered the constant-time implementation provided in BIKE Additional Implementation [DGK20a] with minor changes to account for the updated threshold function in BIKE's last revision [ABB+21].

Notice how δ and N_{Iter} are the same in all security levels. The threshold function is an increasing linear function on the syndrome weight truncated above the minimum value $(d+1)/2$. Since the threshold function is used to determine when to flip a bit, this means that when the weight of the syndrome $\mathbf{s}$ is large, fewer bits will be flipped.

Security level λ	δ	N_{Iter}	Thresh($\mathbf{s}$)	Cycles Portable	Cycles AVX512
128	3	5	$\max(36, \lfloor 0.00697220\text{w}(\mathbf{s}) + 13.5300 \rfloor)$	10955732	1323322
192	3	5	$\max(52, \lfloor 0.00526500\text{w}(\mathbf{s}) + 15.2588 \rfloor)$	32982825	4130087
256	3	5	$\max(69, \lfloor 0.00402312\text{w}(\mathbf{s}) + 17.8785 \rfloor)$	94902236	11497288

Table 5.2: BGF parameters and their corresponding performance when considering the portable and AVX512 implementations.

5.4 Critical analysis of BGF

In this section, we dive a little deeper into the BGF decoding algorithm. This allows us to better understand why BGF is effective, but, more importantly, it will show some

[28] Intel® Xeon™ Gold 5118 CPU at 2.30GHz.

of BGF's weaknesses and lay the ground over which a better decoder can be designed. It is well-known to be difficult to provide a theoretical analysis for QC-MDPC iterative decoders, because of the inherent dependency caused by the circulant matrices involved. Therefore, our analysis is based on observations of BGF's behavior in practice.

5.4.1 BGF's first iteration: The Black-Gray step

Let us first discuss BGF's first iteration and its importance for the extrapolation framework. As described in the previous section, in the first iteration, BGF performs a sequence of 3 bit-flipping calls: one BitFlipIter followed by two BitFlipMaskedIter.

We know that BitFlipIter flips all bits whose UPC counters are above a certain threshold. This makes it very sensible to the threshold selected, as illustrated in Figure 5.2. Consider the difference if, by chance, the threshold $\tau_0 = 76$ was selected, then the number of errors made after calling BitFlipIter, that is, correct bits that would be incorrectly flipped, would be twice the number if $\tau_0 = 77$ were selected.

This problem is particularly important under the extrapolation framework, where the algorithm needs not only to perform well, but also to improve its performance at a very fast rate for small, but increasing, values of r. Therefore, BGF uses a very conservative threshold in the first iteration BitFlipIter. Additionally, the black and gray regretting phases, corresponding to the two calls of BitFlipMaskedIter, also work by flipping a controlled number of bits: only those bits in the black or gray masks whose UPC is above $(d+1)/2$. This makes the whole first iteration very conservative.

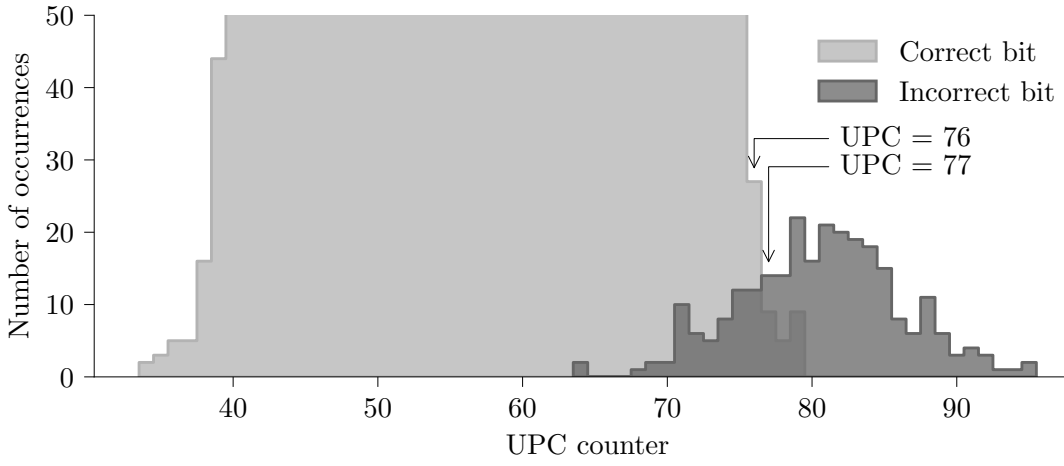

Figure 5.2: Histogram of the UPC counters for each of the $2r$ bits in the partial error vector $\mathbf{e} = \mathbf{0}$, in the beginning of the first iteration, separated by the cases when the bit is right or wrong. The values correspond to a real observation corresponding to the BIKE Level 5 security parameters.

Even though a conservative first iteration is important to ensure a fast DFR decay when r increases, it may result in useless iterations when r is close to the value when negligible DFR is reached. In particular, for the case presented in Figure 5.2, where $r = 40973$, ThresHreturned $\tau_0 = 86$. This would result in no error being made after BitFlipIter,

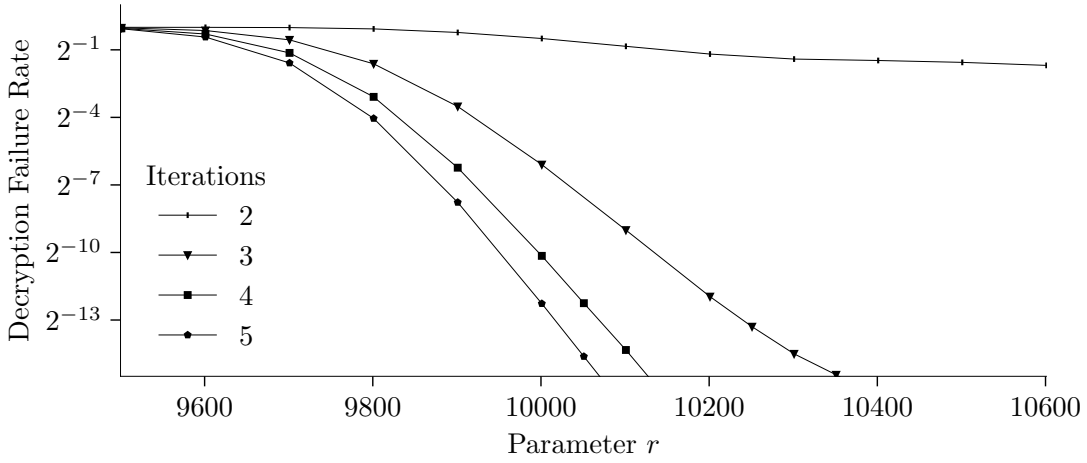

Figure 5.3: The impact of the number of BGF iterations on the DFR, considering parameter set BIKE Level 1 ($t = 134$, $w = 142$).

but at the cost of flipping only a small number of bits, compared to the case where $\tau_0 = 80$, for example.

This suggests that removing the black regret step may be a good starting point for optimization. For example, we could merge both black and gray regret steps into one iteration in such a way that the black regret is critical for small r, but when r gets larger, the gray regret steps gets more important than the black one. This is the key idea behind our PickyFlip iteration that we introduce in Section 5.5.

5.4.2 The number of iterations and the threshold function

One straightforward method to improve the decapsulation performance would be to decrease the number of BGF iterations, at the cost of increasing the key sizes. Intuitively, one may think that there is a direct trade-off between the number of iterations and the block length parameter r: the DFR may not decay as fast when using a lower number of iterations, but one might be lucky to obtain a reasonable value of r after the extrapolation. However, as discussed in the previous section, since the thresholds are so conservative, if the number of iterations is too small, the decoder may not be able to fully correct the errors even for large values of r.

Figure 5.3 shows how the number of iterations affects the decay of the DFR as a function of r. Notice how 2 iterations are not enough to allow for a complete decoding of errors of weight $t = 134$. Furthermore, the curve for 3 iterations does not appear to be concave, therefore it is not safe to use the extrapolation framework for this value. This odd behavior of the DFR curves for 2 and 3 iterations is caused by the following problem. On the one hand, increasing r should make it easier to correct more errors, since there is more redundancy, but, for large values of r, the threshold τ_0 used in the first iteration is so high that only very few errors are corrected in the first iteration.

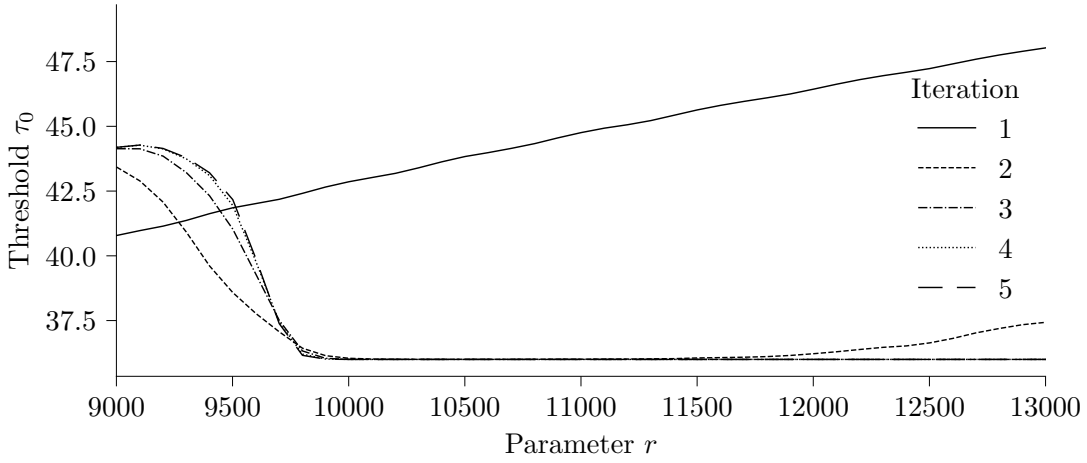

Figure 5.4: Average values of threshold τ_0 in each of the 5 iterations of BGF, considering parameter set BIKE Level 1 ($t = 134$, $w = 142$).

Let us analyze the thresholds τ_0 in more detail. The average values of the thresholds τ_0 used in each of BGF's iteration are shown in Figure 5.4, considering 10000 decapsulations under BIKE Level 1 parameter set. We make three observations. First notice how the first threshold increases as r increases. This is a consequence of the linear dependency of τ_0 on the syndrome weight $\mathrm{w}(\mathbf{s})$, which turns out to increase with r. The second observation is that the thresholds used in iterations 2 to 5 appear to converge to the floor $(d+1)/2 = 36$. This happens because the first iteration, in general, is able to flip a sufficiently large number of errors, and leave only fine adjustments for the next iterations to deal with. The third is that τ_0, for the second iteration, starts increasing after $r = 11500$. This is caused by the threshold in the first iteration being too high, which leaves a lot of errors to be corrected by the second iteration.

5.4.3 Impact of the threshold on the concavity assumption

Back to the DFR curves, the non-concave behavior of the curves for 2 and 3 iterations raises a potentially deep problem with the BGF threshold: why should we expect the curves for 4 and 5 iterations to be concave as well? It is possible that we just cannot see an inflection point because it is located at a DFR smaller than what we can simulate.

To evaluate the concavity of the DFR curves for 5 iterations, we propose the following experiment. Consider BIKE Level 1 parameter set. Since we cannot see the inflection points for $t = 134$, we can exaggerate the error weight t so that we can see the DFR curve in the interval of interest. Ideally, it should be concave at least within all values of $r < 12323$, since this is the extrapolated value of r for BIKE Level 1.

As we can see in Figure 5.5, this is not what happens for $t = 151$, 153 and 155, for BGF with 5 iterations. Therefore, considering our results regarding the non-concavity of BGF with 2 and 3 iterations, together with the non-concavity of BGF with 2 to 5 iterations

when $t = 151$, we believe that it is not conservative to assume that the DFR curve for BGF is concave. We also tested BGF for levels 3 and 5, observing an analogous behavior for $t = 220$ and $t = 300$, respectively.

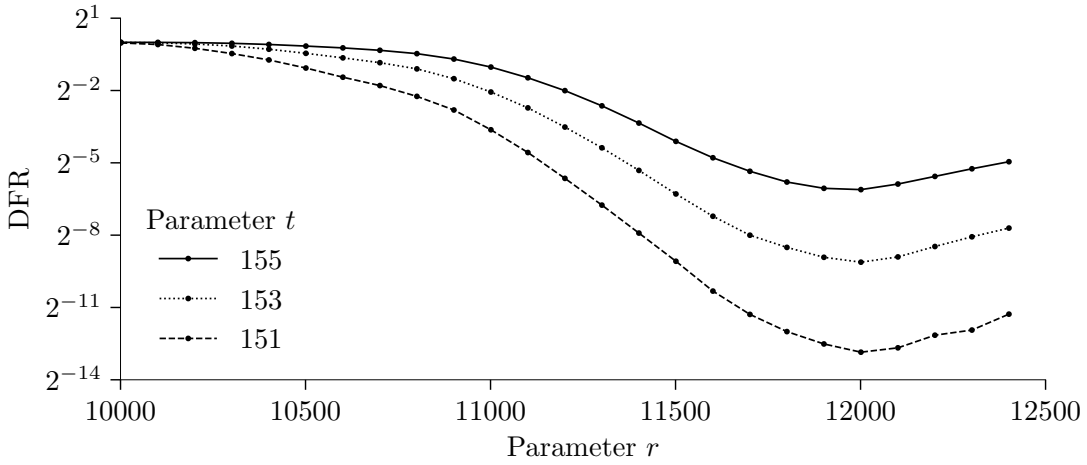

Figure 5.5: The DFR plot for different values of t considering BIKE Level 1 parameter set.

The main cause for this behavior appears to be the threshold function that depends on $\mathrm{w}(\mathbf{s})$. We conclude that it is not safe to use it for the first, and most important, iteration, but Figure 5.4 suggests that it might be used in further iterations, since it converges to $(d+1)/2$. Initially, we though that the threshold problem would be fixed by defining a maximum value for τ_0. In our exploratory tests, this indeed make concave DFR curves for exaggerated values of t, but the error correction was negatively affected. Therefore, we leave the problem of finding better thresholds for future work.

Our approach to deal with the first iteration is simple: we do not use a simple threshold to flip bits. Instead of starting with a BitFlip iteration, we propose to start with FixFlip, a new type of iteration that works by flipping a predetermined number of bits that have the largest corresponding UPC.

5.5 PickyFix

In this section, we describe a new BIKE decoder called PickyFix. Similar to other iterative decoders for LDPC codes, PickyFix works by performing a sequence of iterations that progressively increases the knowledge of the secret sparse error used for encrypting. However, it differs significantly in how it chooses which bits to flip in its iterations. We begin by defining two new types of auxiliary procedures: the FixFlip and PickyFlip iterations, that are the building blocks of our decoder.

5.5.1 The FixFlip auxiliary iteration

While the majority of previous bit-flip approaches are based on flipping all bits whose UPC counters are above a certain threshold, FixFlip flips a predetermined number of bits, denoted by n_{Flips}, that have the highest UPC counters. The formal description of a full iteration of FixFlip is described as Algorithm 5.4.

Algorithm 5.4 The FixFlip iteration.

1: **procedure** FixFlipIter($\mathbf{H}, \mathbf{z}, \mathbf{e}, \mathbf{s}, n_{\text{Flips}}$)
2: $\texttt{upc} \leftarrow \left[\left|\text{supp}\,(\mathbf{s}) \cap \text{supp}\left(\mathbf{H}_j^\top\right)\right| \text{ for } j = 1 \text{ to } 2r\right]$ ▷ We need the full UPC array
3: $\texttt{worst\_indexes} \leftarrow$ list of the indexes j of the n_{Flips} largest values of $\texttt{upc}$
4: **for** j in $\texttt{worst\_indexes}$ **do**
5: $\mathbf{e}_j \leftarrow \overline{\mathbf{e}_j}$ ▷ Flips coordinate j of $\mathbf{e}$
6: $\mathbf{s} \leftarrow \mathbf{z} + \mathbf{e}\mathbf{H}^\top$ ▷ Recomputes the partial syndrome
7: **return** $\mathbf{e}, \mathbf{s}$

Almost every step of the algorithm is standard for other bit-flipping algorithms. However, despite its simplicity, one has to be careful with line 3 when implementing the FixFlip iteration, In Section 5.7 we discuss this issue and show how this can be done efficiently in linear time on r by using important observations on QC-MDPC parameters.

This iteration is very useful at the start of the decoding process, when there is a lot of uncertainty about the correctness of the bits. We can point two immediate advantages of using FixFlip. First, since the number of flips is fixed, the number of wrong flips done by this iteration is limited. This makes FixFlip useful for small values of r, which is an important property for decoders to be used in Vasseur's [Vas21] DFR extrapolation framework. Second, and most important, FixFlip is immune to the problem of BGF's first threshold that gets larger as r grows, since it does not rely on a generic threshold function that depends only on $|\mathbf{s}|$. In fact, the threshold function for FixFlip depends directly on the UPC values and the target number n_{Flips} of bits to flip.

5.5.2 The PickyFlip auxiliary iteration

PickyFlip is very similar to the BitFlip iteration, except that it uses 2 different threshold: τ_{In} is used to flip zeros to ones and τ_{Out} to flip ones to zeros. In particular, PickyFlip requires that the threshold to flip a zero to a one is greater than or equal to the threshold to flip a one to zero. This makes it picky with respect to the support of $\mathbf{e}$ and explains why we use *in* and *out* to differentiate the thresholds. The iteration is formally described as Algorithm 5.5.

The power of this iteration is that the weight of $\mathbf{e}$ does not grow too much in one iteration because it is easier to give up on a 1 in the partial error vector $\mathbf{e}$ than to accept one more. Additionally, the effect of one PickyFlip iteration is similar to the sequence of black regret and gray regret steps, for a sufficiently high r. Luckily, because of its similarity

Algorithm 5.5 The PickyFlip iteration.

procedure PickyFlipIter($\mathbf{H}, \mathbf{z}, \mathbf{e}, \mathbf{s}, \tau_{\text{In}}, \tau_{\text{Out}}$)
  for $j = 1$ to $2r$ **do**
    $\mathtt{upc}_j = \left|\text{supp}\left(\mathbf{s}\right) \cap \text{supp}\left(\mathbf{H}_j^\top\right)\right|$
    if $\mathbf{e}_j = 0$ and $\mathtt{upc}_j \geq \tau_{\text{In}}$ **then**
      $\mathbf{e}_j \leftarrow \overline{\mathbf{e}_j}$
    else if $\mathbf{e}_j = 1$ and $\mathtt{upc}_j \geq \tau_{\text{Out}}$ **then**
      $\mathbf{e}_j \leftarrow \overline{\mathbf{e}_j}$
  $\mathbf{s} \leftarrow \mathbf{z} + \mathbf{e}\mathbf{H}^\top$ ▷ Recomputes the partial syndrome
  return $\mathbf{e}, \mathbf{s}$

with the BitFlip iteration, it can be easily implemented by small adjustments of the code by Drucker et al. [DGK20a] in BIKE Additional Implementation.

5.5.3 The PickyFix decoder

We are now ready to define a full decoder, which is described as Algorithm 5.6. To allow for a direct comparison between PickyFix and BGF, we decided to define it in a similar fashion: the first iteration makes 3 calls of the auxiliary steps, which are then followed by single calls in the next $N_{\text{Iter}} - 1$ iterations.

Algorithm 5.6 The PickyFix decoding algorithm.

procedure PickyFix($\mathbf{H} = [\mathbf{H}_0|\mathbf{H}_1], \mathbf{z} = \mathbf{c}\mathbf{H}_0^\top$)
  $\mathbf{e} \leftarrow \mathbf{0} \in \mathbb{F}_2^{2r}$ ▷ Initializes the partial error vector
  $\mathbf{s} \leftarrow \mathbf{z}$ ▷ Initializes the partial syndrome
  for $i = 1$ to N_{Iter} **do**
    ▷ Every time $\mathbf{e}$ and $\mathbf{s}$ are updated, it holds that $\mathbf{s} = \mathbf{z} + \mathbf{e}\mathbf{H}^\top$
    if $i = 1$ **then**
      $\mathbf{e}, \mathbf{s} \leftarrow$ FixFlipIter($\mathbf{H}, \mathbf{z}, \mathbf{e}, \mathbf{s}, n_{\text{Flips}}$)
      $\mathbf{e}, \mathbf{s} \leftarrow$ PickyFlipIter($\mathbf{H}, \mathbf{z}, \mathbf{e}, \mathbf{s}, \tau_{\text{In}} = \text{Thresh}(\mathbf{s}), \tau_{\text{Out}} = (d+1)/2$)
      $\mathbf{e}, \mathbf{s} \leftarrow$ PickyFlipIter($\mathbf{H}, \mathbf{z}, \mathbf{e}, \mathbf{s}, \tau_{\text{In}} = \text{Thresh}(\mathbf{s}), \tau_{\text{Out}} = (d+1)/2$)
    else
      $\mathbf{e}, \mathbf{s} \leftarrow$ PickyFlipIter($\mathbf{H}, \mathbf{z}, \mathbf{e}, \mathbf{s}, \tau_{\text{In}} = \text{Thresh}(\mathbf{s}), \tau_{\text{Out}} = (d+1)/2$)
  if $\mathbf{e}\mathbf{H}^\top = \mathbf{z}$ **then** ▷ This condition is equivalent to $\mathbf{s} = \mathbf{0}$
    return $\mathbf{e}$
  else
    return $\perp$ ▷ Decoding failure

The threshold τ_{Out} for PickyFix is fixed as $(d+1)/2$ in every iteration, which is the value typically used as the minimum threshold for flipping bits. For the value of τ_{In}, we decided to use the BGF's auxiliary function Thresh which was carefully built by the BIKE team and is sufficiently restrictive for our use case.

FixFlip depends on the following parameters: the number n_{Flips} of flips to be done by

FixFlipIter and the number N_{Iter} of iterations. These parameters depend on the security level and significantly impact the decoder's performance. We analyze these parameters in the next section.

5.6 Analysis

The main problem when searching for good parameters $(n_{\text{Flips}}, N_{\text{Iter}})$ is that they are not independent. For example, if n_{Flips} is too small, we may need a large number N_{Iter} of iterations to compensate. To simplify our search, we will take a greedy approach and break the search into two parts.

In this section, first we find good values for n_{Flips} by focusing only on the first iteration and then show that these values indeed yield decoders with a concave DFR curve. Finally, we proceed to evaluate the decoder performance for different number N_{Iter} of iterations.

5.6.1 Choosing the FixFlip parameter

Intuitively, the best value of n_{Flips} is the one that minimizes the number of errors left to be corrected by further PickyFlip iterations. Ideally, one could see how each possible value of n_{Flips} affects the DFR curves following the extrapolation framework, and choose the one that has the fastest decay. The problem of this approach is that these experiments are very expensive and could easily take months of computing power.

To deal with this problem, instead of counting decoding failures, we count the average number of uncorrected errors left, which can be estimated with a much smaller sample than what is needed for the DFR estimation. Consider the curves $\text{PickyFix}_1^{n_{\text{Flips}}}(r)$ that represent the average number of errors left after the first iteration of PickyFix when the FixFlip iteration performs n_{Flips} bit flips. Similarly, define the curve $\text{BGF}_1(r)$ as the average number of errors left after the first iteration of BGF.

Figure 5.6 shows selected curves, where the average number of errors left was obtained by simulations of 10000 runs. Notice how each PickyFlip curve eventually leaves about 0 errors after the first iteration. Furthermore, we can see that BGF appears to stall its error correction in its first iteration as r increases. In Level 1, BGF even starts to leave more errors for sufficiently large values of r, which is a consequence of the very conservative threshold used in the first iteration that we discuss in Section 5.4.1.

To obtain the best value of n_{Flips} we used the following criteria: for each security level,

Parameter set	Security level	Value r_0	n_{Flips}	$\text{PickyFix}_1^{n_{\text{Flips}}}(r_0)$	$\text{BGF}_1(r_0)$
BIKE Level 1	128	11001	55	0.0	63.97
BIKE Level 3	192	21201	65	0.0	109.06
BIKE Level 5	256	35001	100	0.0	105.79

Table 5.3: The best values of n_{Flips} for each security level. Value r_0 denotes the first value of r when Equation 5.1 is satisfied.

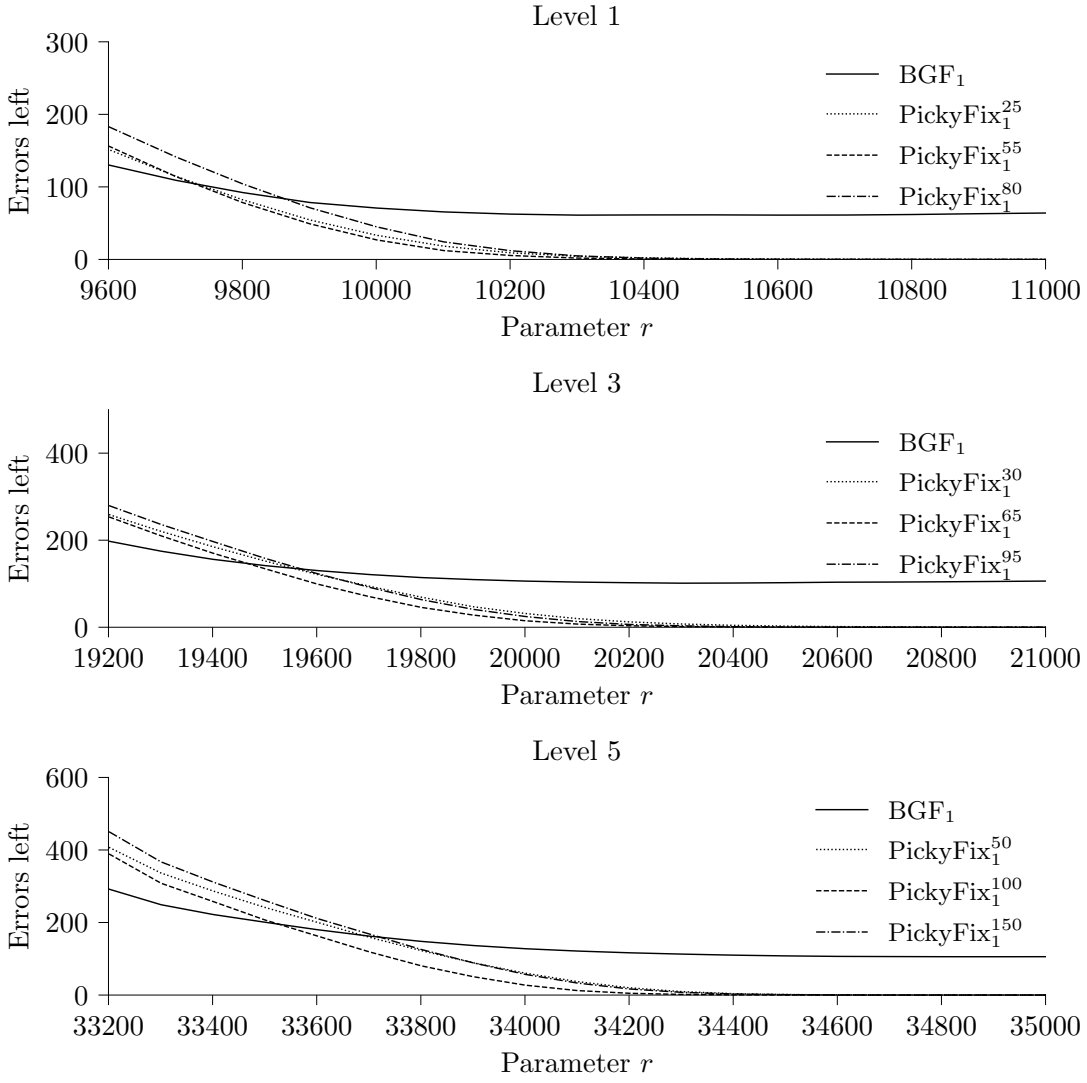

Figure 5.6: Comparison of the number of uncorrected errors after the first iteration for BGF and FixFlip using different values of n_{Flips}, for the three BIKE parameter sets. The different values of n_{Flips} are indicated by the label format $\text{FixFlip}_1^{n_{\text{Flips}}}$.

select the value n_{Flips} such that

$$\text{PickyFix}_1^{n_{\text{Flips}}}(r) = 0, \tag{5.1}$$

for the lowest value of r. Furthermore, we restricted the search for n_{Flips} to multiples of 5 to speed up the search. The best values of n_{Flips} obtained for each security level are shown in Table 5.3, where 10000 tests were performed to estimate $\text{PickyFix}_1^{n_{\text{Flips}}}(r)$ for each r.

Let us now see how PickyFix behaves with respect to the concavity with an experiment similar to the one done in Section 5.4.3 for BGF. First notice that we could not use $t = 155$ because PickyFix was much better than BGF's and its DFR quickly got to the point where no failure could be observed in our simulation. Therefore, we had to consider $t = 160$.

Figure 5.7 shows our results for this experiment. We invite the reader to compare this figure with Figure 5.4.3 and see that, not only PickyFix's DFR appears to be concave in the same interval, but it also outperforms BGF with 5 iterations for a higher value of t. Furthermore, we also tested PickyFix for levels 3 and 5, using $t = 240$ and $t = 330$, respectively, and the DFR curves appear to be concave, unlike the ones for BGF.

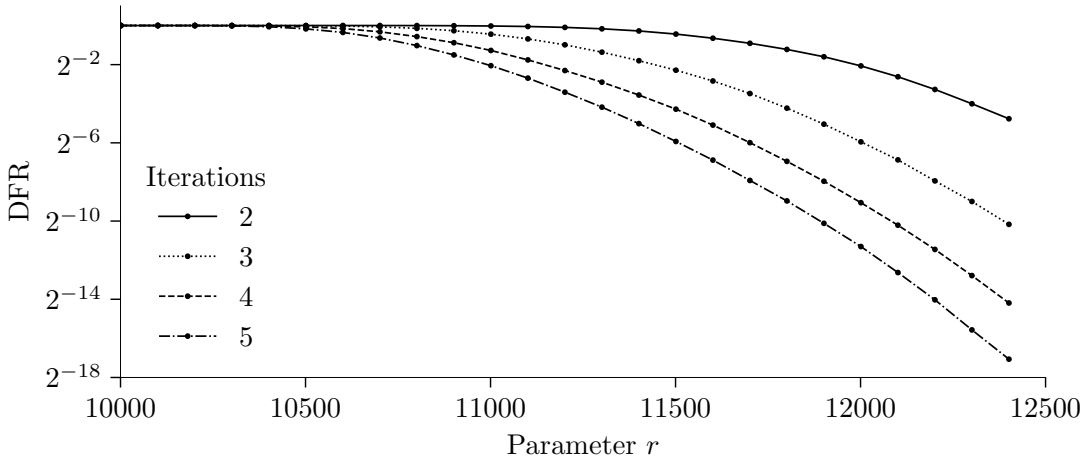

Figure 5.7: The DFR curves for PickyFix when using 2 to 5 iterations considering the BIKE Level 1 parameter set with $t = 160$.

5.6.2 Achieving negligible DFR

Now comes the most important evaluation of PickyFix, which consists of its decoding performance under the extrapolation framework. Our results are shown in Figure 5.8. The number of tests to determine each DFR estimate was selected to be enough to obtain approximately 1000 failures (at least) for each point and can be found in `data/setup/dfr_experiment.csv`.

Table 5.4 shows the results for the DFR extrapolation of the curves considered in Figure 5.8, together with the performance of our constant-time implementations. The extrapolation was done for the last two points (r_A, p_A) and (r_B, p_B) where more than 1000 failures were observed and considered $\alpha = 0.01$ for the Clopper-Pearson method to build the confidence interval for p_A and p_B.

We can see, from Table 5.4, that even with less than 5 iterations, the extrapolated parameter r for each security level does not differ by much from the parameters proposed by the BIKE team using BGF (Table 5.1). However, since PickyFix also works with a reduced number of iterations, its performance can be significantly better, as we can see in Table 5.6.

From the results presented in this section, PickyFix looks like a promising decoder for BIKE. When dealing with QC-MDPC codes, it is also important to consider how decoders perform when using weak keys, which are those that tend to have higher DFRs,

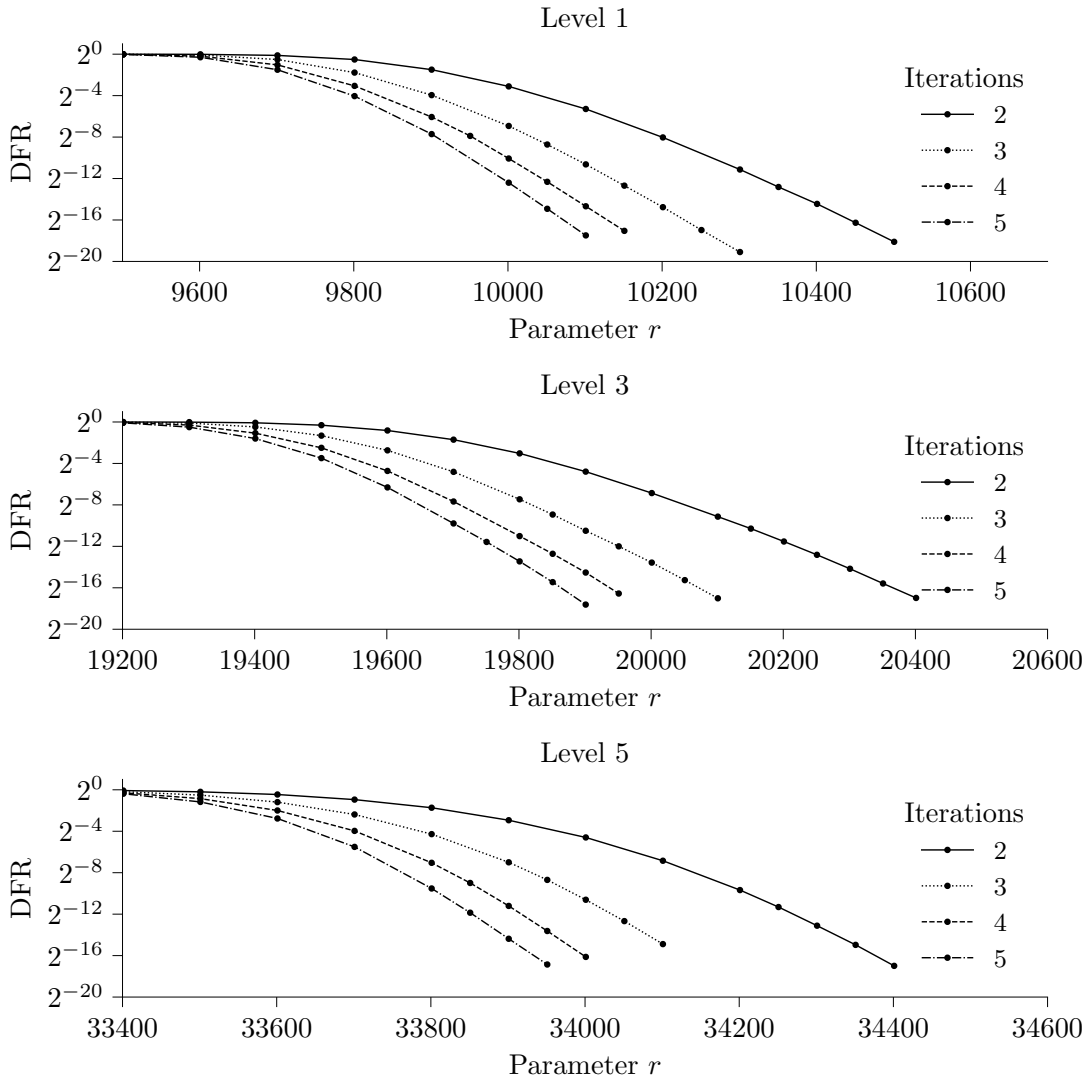

Figure 5.8: The DFR for PickyFix when using 2 to 5 iterations, considering all security levels.

or when decoding near-codeword error patterns, which are error patterns that are more difficult to correct [DGK19, SV20b, Vas21]. While these patterns are known to exist for QC-MDPC codes, they are not yet known to cause issues in the DFR estimation, as mentioned in BIKE's latest revision [ABB+22]. It is both important to better understand their theoretical impact and also to compare PickyFix with other decoders with respect to these corner cases, and this is left for our future work.

The main drawback of the PickyFix is that the FixFlip auxiliary iteration used by PickyFix is inherently more complex than those used by BGF. In the next section, we describe how to efficiently implement PickyFix in constant-time and show that our decoder provides a major speedup over BGF for all security levels.

Security	Iterations	(r_A, p_A)	(r_B, p_B)	r	Relative increase in r
128	2	$(10451, 1.268 \times 10^{-5})$	$(10501, 3.54 \times 10^{-6})$	13829	12.22%
	3	$(10251, 7.78 \times 10^{-6})$	$(10301, 1.788 \times 10^{-6})$	13109	6.38%
	4	$(10101, 3.816 \times 10^{-5})$	$(10151, 7.37 \times 10^{-6})$	12739	3.38%
	5	$(10051, 3.221 \times 10^{-5})$	$(10101, 5.44 \times 10^{-6})$	12413	0.73%
192	2	$(20351, 2.036 \times 10^{-5})$	$(20401, 7.74 \times 10^{-6})$	27397	11.1%
	3	$(20051, 2.546 \times 10^{-5})$	$(20101, 7.56 \times 10^{-6})$	25867	4.9%
	4	$(19901, 4.23 \times 10^{-5})$	$(19951, 1.04 \times 10^{-5})$	25189	2.15%
	5	$(19851, 2.228 \times 10^{-5})$	$(19901, 4.96 \times 10^{-6})$	24677	0.07%
256	2	$(34351, 3.128 \times 10^{-5})$	$(34401, 7.68 \times 10^{-6})$	41411	1.07%
	3	$(34051, 1.532 \times 10^{-4})$	$(34101, 3.294 \times 10^{-5})$	39901	-2.62%
	4	$(33951, 7.91 \times 10^{-5})$	$(34001, 1.398 \times 10^{-5})$	39163	-4.42%
	5	$(33901, 4.704 \times 10^{-5})$	$(33951, 8.42 \times 10^{-6})$	39019	-4.77%

Table 5.4: Results of the DFR extrapolation for BIKE, considering FixFlip with 2 to 5 iterations and multiple security levels. For the relative increase in r, we compared the extrapolated values for the FixFlip decoder with the values of r used by the BGF decoder shown in Table 5.1.

5.7 Efficient implementation in constant time

The efficient constant-time implementation proposed by the BIKE team is based on Chou's[Cho16] QcBits with further improvements by Guimarães et al. [GAB19] and Drucker et al. [DGK20b, DG19]. Using these ideas, Drucker et al. [DGK19, DGK20c] proposed the BGF implementation that is the best performing decoder up to this day, which is implemented in BIKE's Additional Implementation[DGK20a].

We based our PickyFix implementation on Drucker's et al.[DGK20a] code, which implements, in constant-time, most of the procedures required for both PickyFlip and FixFlip iterations. This includes, for example, the syndrome and UPC counters computations, and algorithms to flip bits given a threshold.

This section begins with a high-level description on how to adapt Drucker's et al.[DGK20a] implementation to perform the PickyFlip iteration in constant-time. Then we give a more detailed explanation on how to implement the procedures needed by FixFlip that are significantly different from what is used by previous decoders. We end this section with a performance evaluation of our constant-time implementation, which is available at https://github.com/thalespaiva/pickyfix.

5.7.1 Implementing the PickyFlip iteration

Remember that PickyFlip is similar to the BitFlip iteration, except that it uses a different threshold to flip zeros and ones. More specifically, consider the BitFlipIter described in Algorithm 5.2. Notice how if $\texttt{upc}_j \geq \tau_0$ it inverts $\mathbf{e}_j$, but if $\tau_0 - \delta \leq \texttt{upc}_j < \tau_0$, it updates $\texttt{GrayMask}_j = 1$. BitFlip behavior is then very similar to PickyFix if we let $\tau_0 = \tau_{\text{In}}$ and $\delta = \tau_{\text{In}} - \tau_{\text{Out}}$.

BIKE's efficient implementation of BitFlip is based on QcBits [Cho16], and we imple-

mented PickyFix by reusing their implementation. Since the details of this implementation are already described by Chou [Cho16], we give here only a brief description of how it works.

Suppose we want to flip all bits in $\mathbf{e}$ whose UPC counters are above a threshold τ_{In}. First, all UPC counters are computed in bitsliced form. Since the UPC counters are lower than or equal to $d = w/2$, then $\lceil \log_2(d) \rceil$ slices are enough. Second, the implementation performs a bitsliced subtraction of τ_{In} over all UPC counters. Therefore, the 0 bits in the last slice, which contains the most significant bits, indicate that the UPC was greater than or equal to τ_{In}, and thus the corresponding bit in $\mathbf{e}$ should be flipped.

Notice that PickyFix performs the procedure above two times: one for τ_{In} and other to τ_{Out}. However, the computation of UPC counters, which is the most costly step, is only done once for the two thresholds. The cost of the call is then very similar to the complexity of BitFlipIter.

5.7.2 Implementing the FixFlip iteration

Most of the steps needed by the FixFlip algorithm are common to all variants of the original bit-flipping decoder proposed by Gallager [Gal62]. Therefore, we can base our implementation in the most efficient constant-time implementations of QC-MDPC decoders, if we can efficiently implement the sorting step of FixFlip, corresponding to line 3 of Algorithm 5.4.

Simply put, the main problem we need to solve is: given a list of UPC counters, flip the n_{Flips} bits that have the largest counters. This motivates us to call the set of indexes of entries to be flipped as a FixFlip set, which is formally defined bellow.

Definition 5.7.1 (FixFlip set)**.** Consider a list of UPC counters $U = (u_1, \ldots, u_{2r})$. A FixFlip set S with respect to U and n_{Flips} is a set of n_{Flips} indexes such that $u_i \geq u_s$ for all $i \notin S$ and for all $s \in S$.

Notice that, in general, there are more than 1 FixFlip set for the same list of UPC counters. For example, for a list of UPC counters $U = (3, 5, 2, 3, 7, 1, 3, 1)$ and $n_{\text{Flips}} = 4$, then $S_1 = \{1, 2, 4, 5\}$ and $S_2 = \{1, 2, 5, 7\}$ are two valid FixFlip sets. Furthermore, notice that any FixFlip set S for U can be constructed by the threshold $\tau = 3$ and the integer $n_\tau = 1$ by taking every index i whose UPC is strictly greater than τ and also taking n_τ indexes whose UPC is equal to τ. The pair (τ, n_τ) is then called a FixFlip threshold, and is formally defined next.

Definition 5.7.2 (FixFlip threshold)**.** Let $U = (u_1, \ldots, u_{2r})$ be a list of UPC counters. A pair (τ, n_τ) is a FixFlip threshold with respect to U and n_{Flips} if, for any FixFlip set S can be partitioned into $S = S_{>\tau} \cup S_{=\tau}$ such that $S_{>\tau} = \{s \in S : u_s > \tau\}$, $S_{=\tau} = \{s \in S : u_s = \tau\}$ and $|S_{=\tau}| = n_\tau$.

This notion helps us to reduce the problem of flipping the bits with the largest UPC values to finding a FixFlip threshold, as shown in Algorithm 5.7. The idea of the algorithm

Algorithm 5.7 Algorithm to flip the n_{Flips} entries of $\mathbf{e}$ with largest UPC counters.

```
1:  procedure FlipWorstFitEntries(n_Flips, upc)
2:      τ, n_τ ← FixFlipThreshold(n_Flips, upc)
3:      N_τ ← |{i : upc_i = τ}|
4:      FlipFlagsForThreshold ← Random binary vector of N_τ bits with weight n_τ
5:      η ← 0                          ▷ Counts the number of bits seen whose upc is τ
6:      for i = 1 to 2r do
7:          if upc_i > τ then
8:              e_i = ē_i
9:          else if upc_i = τ then
10:             η ← η + 1
11:             if FlipFlagsForThreshold_η = 1 then
12:                 e_i = ē_i
13:     return e
```

is to flip all bits whose UPC is above τ, and use the array `FlipFlagsForThreshold` to control which set of n_τ bits should be flipped among all of the N_τ bits whose UPC is τ.

The conditionals in Algorithm 5.7 can be implemented in constant-time using condition masks. However, there are two aspects that are important to notice when converting the algorithm to a constant-time implementation. The first is that it is not trivial to implement FixFlipThreshold in constant-time. The second is that, to generate the random vector `FlipFlagsForThreshold` of fixed weight in line 4, and to hide the accesses to index η in line 11, we need a tight upper bound on N_τ. In the next two sections, we describe how our constant-time implementation deals with these concerns.

Computing the FixFlip threshold

The straightforward solution is to use general sorting algorithms, such as quicksort, to sort the indexes based on the corresponding UPC counters' values, and then return the first n_{Flips} indexes. There are two problems with this approach. The first is that the average complexity would be $O(r \log r)$ which would result in an iteration much costlier than that of BGF or BG. The second, and most problematic one, is that the algorithm would not be constant-time and timing attacks would be practical.

Notice that the values of the UPC counters are always in $\{0, \ldots, d\}$, which is a relatively small range, and therefore counting sort is an interesting option that allows for linear sort. The problem with using counting sort in this cryptographic setting is that the constant-time implementation would not be efficient: for every counter, we need to touch all the $d+1$ buckets to avoid cache timing attacks, resulting in $O(wr)$ complexity.

We can do better by analyzing the context in which FixFlip iteration is used. Since the weight t of the error vector is at most $t = 264$, considering security level 5, then it is not necessary to allow for more than 264 flips in each FixFlip iteration. Furthermore, we already saw in Section 5.6.1 that, in practice, n_{Flips} is typically much lower than t for all security levels, and we can safely assume $n_{\text{Flips}} < 256$. This means that, when performing

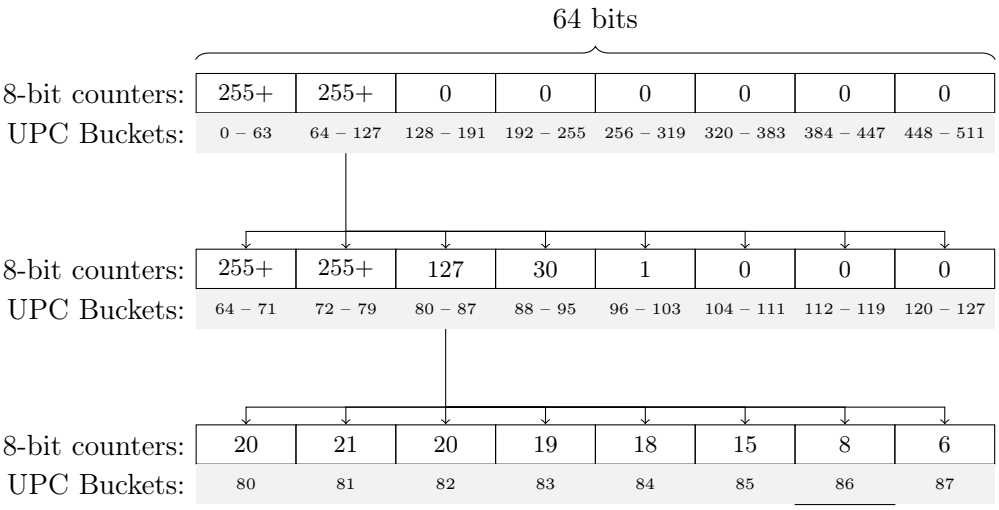

Figure 5.9: Using 3 levels of partial counting sorts to find the FixFlip threshold for $n_{\text{Flips}} = 40$, considering a real execution of the procedure under BIKE Level 5 parameter set. In this example, the FixFlip threshold corresponds to $\tau = 86$ and $n_\tau = 3$.

the counting sort, we only need to count up to 255, since we need only to return the indexes corresponding to the n_{Flips} largest counters. Therefore, 8 bits are needed for each bucket.

Still, even if we can pack 8 buckets into one 64-bit register, we would need to touch all $\lceil (d+1)/8 \rceil$ registers for each counting update. The number of registers would result in $9 \times 2r$ and $18 \times 2r$ operations, considering parameters for levels 1 and 5. But remember that we do not need to count all entries, and we can take what we call the reduced UPC counters approach, which is described next.

Figure 5.9 shows how the algorithm works in a real decoding instance considering BIKE Level 5. Suppose we are given a list $U = (u_1, \ldots, u_{2r})$ of UPC counters and we want to find the FixFlip threshold for U and $n_{\text{Flips}} < 256$. To show our concrete efficient implementation, we assume the following conditions, that hold in the real world parameters.

1. Each UPC counter $u_i \leq d < 512$.

2. The number of bits to flip is $n_{\text{Flips}} < 256$.

The FixFlip threshold is found in 3 counting steps, and each step uses only 8 buckets. For the first step, each bucket i, where i goes from 0 to 7, corresponds to the UPC counters in the interval $[64i, 64i + 63]$. The algorithm then runs from u_1 to u_{2r} counting the occurrences into the buckets, but with the following rule: the counting is done only in 8 bits, and it should not overflow. That is, the maximum count is 255 for each bucket. Now suppose the resulting counts for each bucket is $[255, 255, 0, 0, 0, 0, 0, 0]$, and consider the case $n_{\text{Flips}} = 40$, just like in Figure 5.9. Then the bucket where the FixFlip threshold lives must be Bucket 1, since Buckets 3 to 7 do not have any entry, and there are more than n_{Flips} entries in Bucket 1. Using Bucket $b_1 = 1$ selected in this step, the algorithm proceeds to the next step.

In the second step, the algorithm expands Bucket b_1, and the 8 counting buckets are zeroed. Now, each bucket i will count the UPC counters in the interval $[B_2+8i, B_2+8i+7]$, where $B_2 = 64b_1$. Again, the algorithm runs through the counters in reversed order until it finds where the FixFlip threshold lives. In the case considered in Figure 5.9, Bucket 3 is not enough to contain the threshold since it separates at most 31 UPC counters from the rest. Therefore, the search continues using Bucket $b_2 = 2$.

In the third and last step, Bucket b_2 is expanded, and now each counting bucket will correspond to one UPC value. Formally, each Bucket i will count occurrences of the UPC counter $B_3 + i$, where $B_3 = B_2 + 8b_2$. If we consider the search in Figure 5.9, we can see that it stops at $\tau = 86$, since it has found $6 + 30 + 1 = 37$ UPC values above τ and $n_\tau = 3$ UPC values equal to τ complete the $n_{\text{Flips}} = 40$ bits to be flipped.

Now let us analyze why this algorithm is useful. Since each bucket uses only 8 bits, we can pack all the 8 buckets into a single 64-bits register. Therefore, each update on the counters updates a single register, which avoids the cache-timing attacks. Since 3 rounds are necessary, the threshold is found in about $3 \times 2r$ touches on the counting registers.

Furthermore, let us check that computing the corresponding bucket for an UPC counter is made using constant-time operations. Suppose we want to find the bucket b corresponding to the UPC counter u_i on step ℓ. Then

$$b = \begin{cases} \perp & \text{if } u_i < B_\ell \text{ or } u_i \geq B_\ell + 8^{4-\ell}, \\ \lfloor (u_i - B_\ell)/8^{3-\ell} \rfloor & \text{otherwise.} \end{cases}$$

Both conditions can be evaluated in constant time, since they involve simple unsigned integer comparisons, additions, and the computation of $8^{4-\ell}$ does not involve any secrets. Now for the actual values, if we use 8 bits to represent the buckets, we can let `0xFF` denote the symbol $\perp$. Furthermore, since denominator of the division involving secrets is a power of 8, we can compute $(u_i - B_\ell)/8^{3-\ell}$ in constant time by using a right shift by $3(3 - \ell)$ bits, assuming the processor uses a barrel shifter. This observation is particularly useful when considering the vectorized implementation using AVX512 instructions: the bucket computation can be done in parallel for multiple UPC counters, as they involve simple additions, comparisons and right shifts by a fixed amount.

Generating `FlipFlagsForThreshold` and accessing it in constant time

The generation of a random binary vector of a given weight appears frequently in code-based cryptography. For example, both HQC [MAB+18] and BIKE [ABB+21] itself require such a procedure when generating error vectors or secret keys. There is, however, a key difference between our setup and the constant-weight sampling algorithms used by BIKE: FixFlip must hide both the weight n_τ and the size of the vector N_τ.

Let us first see, in Algorithm 5.8, how the naive Fisher-Yates shuffle works in our case, and then discuss how to make it run in constant-time. We start with a vector of N_τ bits, in which the first n_τ are set to 1 and the rest are set to 0. Then, the algorithm performs n_τ

random swaps to shuffle the first n_τ bits of the array. By the end, if each random integer j generated for the swap is unbiased, then each vector of length N_τ and weight n_τ should be generated with uniform probability $1/\binom{N_\tau}{n_\tau}$.

To implement Algorithm 5.8 in constant-time, we need upper bounds on n_τ, to limit the loops, and on N_τ, to hide the accesses to vector `FlipFlagsForThreshold` when swapping bits in line 9. Notice that, when swapping bits, we only need to hide access to position j, since i is already known in each iteration. Furthermore, notice that we do not use rejection-sampling when selecting the index j because its rejection rate would depend on N_τ. Instead, we use a constant-time modulo reduction of the λ-bit random number, where λ is equal to the security level, to achieve negligible bias.

A trivial upper bound on n_τ is $n_\tau \leq n_{\text{Flips}}$. This allows us to run the loops in lines 3 and 6 in constant time by performing n_{Flips} iterations and using condition masks. Now, to bound N_τ we can focus on the distribution of UPC counters of the wrong bits, that is, those that should be flipped. Let U_τ be the random variable that counts the number of UPC counters, among the wrong bits, that are equal to τ. Notice that, when $N_\tau > 2U_\tau$, then flipping bits whose UPC are equal to τ is more likely to result in a wrong flip. Suppose that we find the smallest value κ, in the interval $0 \leq \kappa \leq t$, such that $\Pr(U_\tau > \kappa) \leq 2^{-\lambda}$, where λ is the security level. Then we only care about flipping bits whose UPC are equal to τ in the case when $N_\tau \leq 2\kappa$, as pointed by the comment in line 4 of Algorithm 5.8.

To find this value κ for each parameter set, we can use Sendrier and Vasseur's [SV19] model for the distributions of UPC counters. Under their model, the UPC counters' distribution for the wrong and right bits are accurately modeled by Binomial distributions with different parameters that are easy to compute. Since we want to consider all possible values of τ, we can search for the smallest κ satisfying the rightmost inequality

$$\Pr[U_\tau > \kappa] \leq \sum_{0 \leq \theta \leq w/2} \Pr[U_\theta > \kappa] \leq 2^{-\lambda},$$

where the distribution of each U_θ is computed using Sendrier and Vasseur's [SV19] model.

Table 5.5 shows the upper bounds on N_τ that we found for each security level. Our

Algorithm 5.8 Generate a random vector of fixed weight using the Fisher-Yates algorithm.

1: **procedure** GenVectorOfFixedWeight(n_τ, N_τ)
2: `FlipFlagsForThreshold` $\leftarrow \mathbf{0} \in \mathbb{F}_2^{N_\tau}$
3: **for** $i = 1$ to n_τ **do**
4: ▷ In the constant-time implementation, bit i is set to 1 only if $N_\tau \leq 2\kappa$
5: `FlipFlagsForThreshold`$_i \leftarrow 1$
6: **for** $i = 1$ to n_τ **do**
7: $u \leftarrow$ Random number of λ bits
8: $j \leftarrow i + (u \mod (N_\tau - i + 1))$ ▷ j is a random integer in range $i \leq j \leq N_\tau$
9: Swap bits i and j of `FlipFlagsForThreshold`
10: **return** `FlipFlagsForThreshold`

implementation uses an array of 64-bit integers to represent `FlipFlagsForThreshold`, and the total number of 64-bit blocks required for 2κ bits is shown in the last column. Notice that, for security levels 128 and 192, it is possible to simultaneously compute N_τ and the FixFlip threshold, since $2\kappa < 255$. To compute κ, we consider the smallest values of r achieving each security level λ, which are taken from Table 5.6. This is a conservative approach, since κ gets smaller for higher r within a fixed security level.

Security level λ	r	w	t	κ	$\Pr(U_\tau > \kappa)$	Upper bound 2κ on N_τ	Number of 64-bit blocks
128	12413	142	134	73	$< 2^{-128.69}$	146	3
192	24677	206	199	103	$< 2^{-195.54}$	206	4
256	39019	274	264	130	$< 2^{-259.13}$	260	5

Table 5.5: Upper bounds on N_τ.

5.7.3 Performance evaluation

We now evaluate the decoder with respect to the full decapsulation time[29], when using PickyFix as a subroutine. For this test, we considered the constant-time implementations of BIKE decapsulation using BGF from BIKE Additional Implementation [DGK20a] and our constant-time PickyFix implementation over their code.

The algorithms are implemented in two modes: the portable implementation and the accelerated one using AVX512 instructions. The testing platform consists of an Intel® Xeon™ Gold 5118 CPU at 2.30GHz. Notice that the decoding step is the most important part of the decapsulation. In our setup, the decoding step consists of 90% of the decapsulation, for the portable implementation, and between 80% and 90%, for the AVX512 implementation[30].

Table 5.6 shows the performance of our constant-time implementation of PickyFix. The basis for the speedup comparison over BGF comes from Table 5.2, for the corresponding security levels. Notice how PickyFix provides major speedups with respect to BGF for all security levels for one very important reason: it can work with a smaller number of iterations. Even if parameter r suffers a slight increase when using only 2 iterations, between 1% ($\lambda = 256$) and 14% ($\lambda = 128$), this is compensated by speedups from 1.47 to 1.18, correspondingly.

[29] This includes the hashes computations required by the FO$^{\not\perp}$ transformation.
[30] These number are considering the PickyFix or BGF decoder with 2 iterations.

Security level	Iterations	r	Portable		AVX512	
			Cycles	Speedup	Cycles	Speedup
128	2	13829	9088162	1.21	1117958	1.18
	3	13109	10244537	1.07	1221821	1.08
	4	12739	11465955	0.96	1327721	1.00
	5	12413	12859593	0.85	1442976	0.92
192	2	27397	25221598	1.31	3196844	1.29
	3	25867	28879874	1.14	3577988	1.15
	4	25189	32580637	1.01	3935606	1.05
	5	24677	36715044	0.90	4350719	0.95
256	2	41411	65388610	1.45	7843855	1.47
	3	39901	76892364	1.23	8928026	1.29
	4	39163	87393964	1.09	10136052	1.13
	5	39019	99706189	0.95	11494770	1.00

Table 5.6: The performance of PickyFix considering the parameters achieving negligible failure rate for each security level. Both the portable and AVX512 implementations were considered, and the speedup is computed with respect to BGF for each level.

5.8 Conclusion and future work

The evidence provided in this paper suggests that PickyFix outperforms BGF both with respect to security and performance. Moreover, we show how PickyFix can be efficiently implemented in constant-time. The only drawback appears to be that the implementation of FixFlip, one of PickyFix's auxiliary iterations, is more involved than that of simple bit-flipping algorithms.

There are several directions one may take to extend this work. It would be interesting to perform a broader exploration of the thresholds used by PickyFlip. For example, to consider looser thresholds for rejecting or accepting ones. On the FixFlip side, notice that we tried to be as general as possible in our implementation. However it may be possible to make it simpler and faster by using the fact that FixFlip is used only in the first iteration. Therefore, one could use statistical analysis to limit the range in which the FixFlip threshold should be searched.

It would be fascinating to see if our implementation of FixFlip can be used to compute better and more complex thresholds. For example, one could use the partial counting of UPC counters to compute thresholds based on the separation of the distributions of UPC for right and wrong bits. On the security side, it is important to understand how PickyFix compares with other decoders in corner cases, such as when using weak keys or decoding near-codeword error patterns [DGK19, SV20b, Vas21]. Finally, it may be interesting to evaluate PickyFix as a decoder for low-density parity-check (LDPC) codes [Gal62].

Acknowledgments

We thank the Continuous Optimization group of Unicamp that provided access to their computer lab to perform the experiments. The lab is supported by FAPESP grant 2018/24293-0. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. This research is part of the INCT of the Future Internet for Smart Cities funded by CNPq proc. 465446/2014-0, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, FAPESP proc. 14/50937-1, and FAPESP proc. 15/24485-9.

Chapter 6

Discussion

This brief concluding chapter consists of a discussion on the potential impact of the results presented in this dissertation. It is also a good opportunity to review some new related results in the last few years, after some our results were published.

Our timing attack on HQC was published in 2019 [PT20] by the same time Wafo-Tapa et al. [WTBBG19] also presented a similar attack. In 2020, Schamberger et al. [SRSWZ20] showed a power-based side-channel attack against HQC and, later that year, Guo et al. [GJ20] showed a more dangerous decryption failure attack on HQC. Since its October 2020 revision, HQC [MAB+18] changed the error correction code from the combination of BCH and repetition codes to a combination of Reed-Muller (RM) and Reed-Solomon codes (RS), since the latter pair of codes yields more efficient parameters. After this change, two power-based side-channel attacks targeting the RM-RS variant of HQC were published [SHR+22, GLG22]. The main limitation of all these power-based attacks is that they, following Wafo-Tapa et al. [WTBBG19] approach, are only applicable by choosing ciphertexts, and therefore detection mechanisms such as the ones proposed by Ravi et al. [RCB22] may thwart the attack. It would then be interesting to see how our approach on the HQC attack behaves in a power side-channel attack, since we only use valid ciphertexts.

In 2022, NIST opened a new request for post-quantum signature proposals. It is believed that a number of new candidates that appeared after 2016 will be submitted, such as PKP-DSS [BFK+19]. Although our attack [PT21] on the permuted kernel problem (PKP) does not directly targets PKP-DSS, it may have an important impact on alternative signature candidates based on PKP: we showed that, when instantiated over the binary field, PKP less secure. Furthermore, the attack may be extended for small fields, which is an important line of future work. It is also important to understand possible tradeoffs between efficiency and security of PKP-based signatures when the underlying field has varying sizes, as small fields are often more friendly for devices with limited resources, such as embedded systems.

With respect to our contribution to BIKE decoding, the PickyFix decoder [BPT22], there are several interesting lines of work that are discussed in the concluding section of Chapter 5. Since the paper was published recently, there were not many developments on

this matter up to this day. It would be very important to see more effort in the theoretical analysis of the decoding process, and, in particular, to strengthen the concavity assumption. There is also room for improvements of PickyFix and also when considering variants of the decoder. Furthermore, if may be of particular interest for designers of decoding algorithms to have a framework for automatic comparison between decoders.

References

[AAC+22] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Carl Miller, Dustin Moody, Rene Peralta, et al. Status report on the third round of the NIST post-quantum cryptography standardization process. *US Department of Commerce, NIST*, 2022. 2

[ABB+17a] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, Arthur Blot, Benjamin Grégoire, Vincent Laporte, Tiago Oliveira, Hugo Pacheco, Benedikt Schmidt, and Pierre-Yves Strub. Jasmin: High-assurance and high-speed cryptography. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1807–1823, 2017. 22

[ABB+17b] Nicolas Aragon, Paulo S. L. M. Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, and Gilles Zémor. BIKE: Bit flipping key encapsulation, 2017. https://bikesuite.org/files/BIKE.2017.11.30.pdf. 80

[ABB+19] Jean-Philippe Aumasson, Daniel J Bernstein, Ward Beullens, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, et al. Sphincs. 2019. 3, 18

[ABB+21] Nicolas Aragon, Paulo S. L. M. Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Santosh Ghosh, Shay Gueron, Tim Güneysu, Carlos Aguilar-Melchor, Rafael Misoczki, Edoardo Persichetti, Jan Richter-Brockmann, Nicolas Sendrier, Jean-Pierre Tillich, Valentin Vasseur, and Gilles Zémor. BIKE: Bit flipping key encapsulation, 2021. https://bikesuite.org/files/v4.2/BIKE_Spec.2021.09.29.1.pdf. 3, 4, 27, 79, 80, 81, 83, 87, 102

[ABB+22] Nicolas Aragon, Paulo S. L. M. Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Santosh Ghosh, Shay Gueron, Tim Güneysu, Carlos Aguilar-Melchor, Rafael Misoczki, Edoardo Persichetti, Jan Richter-Brockmann, Nicolas Sendrier, Jean-Pierre Tillich, Valentin Vasseur, and Gilles Zémor. BIKE: Bit flipping key encapsulation, 2022. https://bikesuite.org/files/v5.0/BIKE_Spec.2022.10.10.1.pdf. 97

[ABD+19] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber algorithm specifications and supporting documentation. *NIST PQC Round*, 2(4), 2019. 3

[ACC+17] Reza Azarderakhsh, Matthew Campagna, Craig Costello, LD Feo, Basil Hess, Amir Jalali, David Jao, Brian Koziel, Brian LaMacchia, Patrick Longa, et al.

Supersingular isogeny key encapsulation. *Submission to the NIST Post-Quantum Standardization project*, 152:154–155, 2017. 3

[ACP+18] Martin Albrecht, Carlos Cid, Kenneth G Paterson, Cen Jung Tjhai, and Martin Tomlinson. NTS-KEM, 2018. 3, 29

[AFS05] Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. A family of fast syndrome based cryptographic hash functions. In *International Conference on Cryptology in Malaysia*, pages 64–83. Springer, 2005. 11

[Ale03] Michael Alekhnovich. More on average case vs approximation complexity. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 298–307. IEEE, 2003. 27

[AMBD+18] Carlos Aguilar-Melchor, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. Efficient encryption from random quasi-cyclic codes. *IEEE Transactions on Information Theory*, 64(5):3927–3943, 2018. xiii, 27, 30, 34, 35

[APRS20] Daniel Apon, Ray Perlner, Angela Robinson, and Paolo Santini. Cryptanalysis of LEDACrypt. In *Annual International Cryptology Conference*, pages 389–418. Springer, 2020. 2, 3

[ASK07] Onur Acıiçmez, Werner Schindler, and Çetin K Koç. Cache based remote timing attack on the AES. In *Cryptographers' track at the RSA conference*, pages 271–286. Springer, 2007. 21, 23

[Bal14] Marco Baldi. *QC-LDPC Code-Based Cryptosystems*, pages 91–117. Springer International Publishing, Cham, 2014. 3, 29

[BB05] David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005. 21

[BBD09] Daniel J Bernstein, Johannes Buchmann, and Erik Dahmen. *Post-quantum cryptography*. Springer Science & Business Media, 2009. 1

[BBGM19] S. Bettaieb, L. Bidoux, P. Gaborit, and E. Marcatel. Preventing timing attacks against RQC using constant time decoding of Gabidulin codes. In *International Workshop on Post-Quantum Cryptography*, 2019. 30

[BCCG92] Thierry Baritaud, Mireille Campana, Pascal Chauvaud, and Henri Gilbert. On the security of the permuted kernel identification scheme. In *Annual International Cryptology Conference*, pages 305–311. Springer, 1992. 54

[BCGM07] Mario Baldi, Franco Chiaraluce, Rene Garello, and Francesco Mininni. Quasi-cyclic low-density parity-check codes in the McEliece cryptosystem. In *Communications, 2007. ICC'07. IEEE International Conference on*, pages 951–956. IEEE, 2007. 27

[BCGO09] Thierry P Berger, Pierre-Louis Cayrel, Philippe Gaborit, and Ayoub Otmani. Reducing key length of the McEliece cryptosystem. In *Progress in Cryptology–AFRICACRYPT 2009*, pages 77–97. Springer, 2009. 27

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Annual international cryptology conference*, pages 1–15. Springer, 1996. 11

[BCL+19] Daniel J Bernstein, Tung Chou, Tanja Lange, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Peter Schwabe, Jakub Szefer, and Wen Wang. Classic McEliece: conservative code-based cryptography. 2019. 3, 27, 29

[BDPA13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 313–314. Springer, 2013. 10

[Ber73] Elwyn R Berlekamp. Goppa codes. *Information Theory, IEEE Transactions on*, 19(5):590–592, 1973. 26

[Ber19] Daniel J. Bernstein. djbsort. https://sorting.cr.yp.to/index.html, 2019. 22

[Beu22] Ward Beullens. Breaking Rainbow takes a weekend on a laptop. *Cryptology ePrint Archive*, 2022. 2

[BFK+19] Ward Beullens, Jean-Charles Faugère, Eliane Koussa, Gilles Macario-Rat, Jacques Patarin, and Ludovic Perret. PKP-based signature scheme. In *International Conference on Cryptology in India*, pages 3–22. Springer, 2019. 4, 20, 51, 54, 55, 107

[BGG+17] Paulo S. L. M. Barreto, Shay Gueron, Tim Gueneysu, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, and Jean-Pierre Tillich. CAKE: Code-based algorithm for key encapsulation. Cryptology ePrint Archive, Report 2017/757, 2017. https://ia.cr/2017/757. 85

[BHH+19] Nina Bindel, Mike Hamburg, Kathrin Hövelmanns, Andreas Hülsing, and Edoardo Persichetti. Tighter proofs of CCA security in the quantum random oracle model. In *Theory of Cryptography Conference*, pages 61–90. Springer, 2019. 2

[BLP11] Daniel J Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents: ball-collision decoding. In *Annual Cryptology Conference*, pages 743–760. Springer, 2011. 56

[BMVT78] Elwyn R Berlekamp, Robert J McEliece, and Henk CA Van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386, 1978. 26, 31

[BPT22] Thales B. Paiva and Routo Terada. Faster constant-time decoder for MDPC codes and applications to BIKE KEM. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022:1–25, Jun. 2022. 107

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993. 11

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures-how to sign with RSA and Rabin. In *International conference on the theory and applications of cryptographic techniques*, pages 399–416. Springer, 1996. 17

[CCJ+16] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. *Report on post-quantum cryptography*. US Department of Commerce, National Institute of Standards and Technology, 2016. 2, 51

[CD22] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH (preliminary version). *Cryptology ePrint Archive*, 2022. 2, 3

[CFSV04] Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence*, 26(10):1367–1372, 2004. 57

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 209–218, 1998. 12

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *Journal of the ACM (JACM)*, 51(4):557–594, 2004. 12

[Cho16] Tung Chou. QcBits: constant-time small-key code-based cryptography. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 280–300. Springer, 2016. 98, 99

[CLS06] Scott Contini, Arjen K Lenstra, and Ron Steinfeld. VSH, an efficient and provable collision-resistant hash function. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 165–182. Springer, 2006. 11

[CS03] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003. 13

[Den03] Alexander W Dent. A designer's guide to KEMs. In *IMA International Conference on Cryptography and Coding*, pages 133–151. Springer, 2003. 13

[DG19] Nir Drucker and Shay Gueron. A toolbox for software optimization of QC-MDPC code-based cryptosystems. *Journal of Cryptographic Engineering*, 9(4):341–357, 2019. 98

[DGK19] Nir Drucker, Shay Gueron, and Dusan Kostic. On constant-time QC-MDPC decoding with negligible failure rate. *IACR Cryptol. ePrint Arch.*, 2019:1289, 2019. 80, 81, 97, 98, 105

[DGK20a] Nir Drucker, Shay Gueron, and Dusan Kostic. BIKE Additional Implementation, 2020. https://bikesuite.org/files/round2/add-impl/BIKE_Additional.2020.02.09.zip. 87, 93, 98, 104

[DGK20b] Nir Drucker, Shay Gueron, and Dusan Kostic. Fast polynomial inversion for post quantum QC-MDPC cryptography. In *International Symposium on Cyber Security Cryptography and Machine Learning*, pages 110–127. Springer, 2020. 98

[DGK20c] Nir Drucker, Shay Gueron, and Dusan Kostic. QC-MDPC decoders with several shades of gray. In *International Conference on Post-Quantum Cryptography*, pages 35–50. Springer, 2020. 4, 80, 83, 85, 98

[DKL+18] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 238–268, 2018. 3

[DR99] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. 1999. 9, 13

[DTVV19] Jan-Pieter D'Anvers, Marcel Tiepelt, Frederik Vercauteren, and Ingrid Verbauwhede. Timing attacks on error correcting codes in post-quantum secure schemes. Cryptology ePrint Archive, Report 2019/292, 2019. https://eprint.iacr.org/2019/292. 30

[EJ01] Donald E. Eastlake 3rd and Paul Jones. US Secure Hash Algorithm 1 (SHA1). RFC 3174, September 2001. 10

[ELPS18] Edward Eaton, Matthieu Lequesne, Alex Parent, and Nicolas Sendrier. QC-MDPC: a timing attack and a CCA2 KEM. In *International Conference on Post-Quantum Cryptography*, pages 47–76. Springer, 2018. 30

[FHK+18] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-fourier lattice-based compact signatures over NTRU. *Submission to the NIST's post-quantum cryptography standardization process*, 36(5), 2018. 3

[FHS+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*, pages 51–68. Springer, 2017. 29, 80

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Annual International Cryptology Conference*, pages 537–554. Springer, 1999. 11, 15, 84

[FOP+16] Jean-Charles Faugere, Ayoub Otmani, Ludovic Perret, Frédéric De Portzamparc, and Jean-Pierre Tillich. Structural cryptanalysis of McEliece schemes with compact keys. *Designs, Codes and Cryptography*, 79(1):87–112, 2016. 27

[FOPT10] Jean-Charles Faugere, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. Algebraic cryptanalysis of McEliece variants with compact keys. In *Advances in Cryptology–Eurocrypt 2010*, pages 279–298. Springer, 2010. 27

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 186–194. Springer, 1986. 11, 18, 51

[FS09] Matthieu Finiasz and Nicolas Sendrier. Security bounds for the design of code-based cryptosystems. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, pages 88–105, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. 56, 61

[Gab85] Ernest Mukhamedovich Gabidulin. Theory of codes with maximum rank distance. *Problemy Peredachi Informatsii*, 21(1):3–16, 1985. 30

[Gab05] Philippe Gaborit. Shorter keys for code based cryptography. In *Proceedings of the 2005 International Workshop on Coding and Cryptography (WCC 2005)*, pages 81–91, 2005. 27

[GAB19] Antonio Guimarães, Diego F. Aranha, and Edson Borin. Optimized implementation of QC-MDPC code-based cryptography. *Concurrency and Computation: Practice and Experience*, 31(18):e5089, 2019. 98

[Gal62] Robert Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962. 81, 85, 99, 105

[Geo92] Jean Georgiades. Some remarks on the security of the identification scheme based on permuted kernels. *Journal of Cryptology*, 5(2):133–137, 1992. 54

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979. 3, 53, 56, 57

[GJ20] Qian Guo and Thomas Johansson. A new decryption failure attack against HQC. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 353–382. Springer, 2020. 107

[GJL15] Qian Guo, Thomas Johansson, and Carl Löndahl. A new algorithm for solving Ring-LPN with a reducible polynomial. *IEEE Transactions on Information Theory*, 61(11):6204–6212, 2015. 35

[GJS16] Qian Guo, Thomas Johansson, and Paul Stankovski. A key recovery attack on MDPC with CCA security using decoding errors. In *22nd Annual International Conference on the Theory and Applications of Cryptology and Information Security (ASIACRYPT)*, 2016. xi, xv, 29, 30, 32, 33, 36, 41, 42, 46, 47, 48, 80, 84

[GKL15] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin backbone protocol: Analysis and applications. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 281–310. Springer, 2015. 11

[GLG22] Guillaume Goy, Antoine Loiseau, and Philippe Gaborit. A new key recovery side-channel attack on HQC with chosen ciphertext. In *International Conference on Post-Quantum Cryptography*, pages 353–371. Springer, 2022. 107

[GMR19] Shafi Goldwasser, Silvio Micali, and Chales Rackoff. The knowledge complexity of interactive proof-systems. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 203–225. 2019. 18

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *Journal of the ACM (JACM)*, 38(3):690–728, 1991. 20

[Gop70] Valerii Denisovich Goppa. A new class of linear correcting codes. *Problemy Peredachi Informatsii*, 6(3):24–30, 1970. 26

[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*, pages 212–219. ACM, 1996. 1, 10

[GT11] Michael T Goodrich and Roberto Tamassia. *Introduction to computer security*. Pearson London, UK, 2011. 8

[HCAL15] Armando Faz Hernández, Roberto Cabral, Diego F Aranha, and Julio López. Implementaçao eficiente e segura de algoritmos criptográficos. *Sociedade Brasileira de Computação*, 2015. 22

[HE11] Tony Hansen and Donald E. Eastlake 3rd. US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF). RFC 6234, May 2011. 10

[HHK17a] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 341–371, Cham, 2017. Springer International Publishing. 2, 11, 15, 16

[HHK17b] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In *Theory of Cryptography Conference*, pages 341–371. Springer, 2017. 35, 37, 80, 82, 84

[JFB+22] Jan Jancar, Marcel Fourné, Daniel De Almeida Braga, Mohamed Sabt, Peter Schwabe, Gilles Barthe, Pierre-Alain Fouque, and Yasemin Acar. "they're not that hard to mitigate": What cryptographic library developers think about timing attacks. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 632–649. IEEE, 2022. 21

[JJ01] Éliane Jaulmes and Antoine Joux. Cryptanalysis of PKP: a new approach. In *International Workshop on Public Key Cryptography*, pages 165–172. Springer, 2001. 54

[JK95] Laurie L Joiner and John J Komo. Decoding binary BCH codes. In *Proceedings IEEE Southeastcon'95. Visualize the Future*, pages 67–73. IEEE, 1995. 30

[JL01] Antoine Joux and Reynald Lercier. "Chinese & Match", an alternative to Atkin's "Match and Sort" method used in the SEA algorithm. *Mathematics of computation*, 70(234):827–836, 2001. 54

[JMM+22] David Joseph, Rafael Misoczki, Marc Manzano, Joe Tricot, Fernando Dominguez Pinuaga, Olivier Lacombe, Stefan Leichenauer, Jack Hidary, Phil Venables, and Royal Hansen. Transitioning organizations to post-quantum cryptography. *Nature*, 605(7909):237–243, 2022. 1

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual international cryptology conference*, pages 388–397. Springer, 1999. 21

[KL21] Jonathan Katz and Yehuda Lindell. *Introduction To Modern Cryptography*. CRC Press/Taylor & Francis Group, 2021. 7, 10

[KMRP19] Eliane Koussa, Gilles Macario-Rat, and Jacques Patarin. On the complexity of the Permuted Kernel Problem. *IACR Cryptology ePrint Archive*, 2019:412, 2019. xii, xiii, 52, 54, 55, 75, 76

[Koc96] Paul C Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Annual International Cryptology Conference*, pages 104–113. Springer, 1996. 21

[Lam79] Leslie Lamport. Constructing digital signatures from a one way function. 1979. 18

[Lan10] Adam Langley. ctgrind – Checking that functions are constant time with valgrind. https://github.com/agl/ctgrind, 2010. 21

[LJS+16] Carl Löndahl, Thomas Johansson, Masoumeh Koochak Shooshtari, Mahmoud Ahmadian-Attari, and Mohammad Reza Aref. Squaring attacks on McEliece public-key cryptosystems using quasi-cyclic codes of even dimension. *Designs, Codes and Cryptography*, 80(2):359–377, 2016. 35, 84

[LKO+21] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. PLATYPUS: Software-based power side-channel attacks on x86. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 355–371. IEEE, 2021. 5

[LLZ+18] Xianhui Lu, Yamin Liu, Zhenfei Zhang, Dingding Jia, Haiyang Xue, Jingnan He, Bao Li, Kunpeng Wang, Zhe Liu, and Hao Yang. LAC: Practical Ring-LWE based public-key encryption with byte-level modulus. Cryptology ePrint Archive, Report 2018/1009, 2018. https://eprint.iacr.org/2018/1009. 30, 49

[LP11] Rodolphe Lampe and Jacques Patarin. Analysis of some natural variants of the PKP algorithm. *IACR Cryptology ePrint Archive*, 2011:686, 2011. 3, 4

[LP12] Rodolphe Lampe. and Jacques Patarin. Analysis of some natural variants of the PKP algorithm. In *Proceedings of the International Conference on Security and Cryptography - Volume 1: SECRYPT, (ICETE 2012)*, pages 209–214. INSTICC, SciTePress, 2012. 4, 20, 53, 54, 55, 70

[M. 12] M. Albrecht and G. Bard. *The M4RI Library – Version 20121224*. The M4RI Team, 2012. 69

[MAB+18] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loıc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, and INSA-CVL Bourges. Hamming quasi-cyclic (HQC). Technical report, Technical report, National Institute of Standards and Technology, 2018. 3, 4, 27, 29, 30, 36, 102, 107

[MB09] Rafael Misoczki and Paulo S. L. M. Barreto. Compact McEliece keys from goppa codes. In *Selected Areas in Cryptography*, pages 376–392. Springer, 2009. 27

[McE78] R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. *Deep Space Network Progress Report*, 44:114–116, 1978. 2, 3, 24, 26, 27, 29, 56

[Mil86] V.S. Miller. Use of elliptic curves in cryptography. *Advances in Cryptology (CRYPTO85)*, pages 417–426, 1986. 1

[Mos18] Michele Mosca. Cybersecurity in an era with quantum computers: will we be ready? *IEEE Security & Privacy*, 16(5):38–41, 2018. 1

[MTSB13] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo S. L. M. Barreto. MDPC-McEliece: New McEliece variants from moderate density parity-check codes. In *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*, pages 2069–2073. IEEE, 2013. 3, 27, 29, 76, 79, 81

[Nie86] Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory*, 15(2):159–166, 1986. 26, 79

[oST16] National Institute of Standards and Technology. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process, 2016. 21

[OTD10] Ayoub Otmani, Jean-Pierre Tillich, and Léonard Dallot. Cryptanalysis of two McEliece cryptosystems based on quasi-cyclic codes. *Mathematics in Computer Science*, 3(2):129–140, 2010. 27

[Pai17] Thales Bandiera Paiva. *Melhorando o ataque de reação contra o QC-MDPC McEliece*. PhD thesis, Universidade de São Paulo, 2017. 41, 45

[PC93] Jaques Patarin and Pascal Chauvaud. Improved algorithms for the permuted kernel problem. In *Annual International Cryptology Conference*, pages 391–402. Springer, 1993. 54, 55

[Por18] Thomas Pornin. Why constant-time crypto? https://www.bearssl.org/constanttime.html, 2018. 22

[Pou97] Guillaume Poupard. A realistic security analysis of identification schemes based on combinatorial problems. *European transactions on telecommunications*, 8(5):471–480, 1997. 54, 55

[Pra62] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962. 35

[PT20] Thales Bandiera Paiva and Routo Terada. A timing attack on the HQC encryption scheme. In Kenneth G. Paterson and Douglas Stebila, editors, *Selected Areas in Cryptography – SAC 2019*, pages 551–573, Cham, 2020. Springer International Publishing. 107

[PT21] Thales Bandiera Paiva and Routo Terada. Cryptanalysis of the binary permuted kernel problem. In *International Conference on Applied Cryptography and Network Security*, pages 396–423. Springer, 2021. 107

[Pub99] FIPS Pub. Data Encryption Standard (DES). *FIPS PUB*, pages 46–3, 1999. 9

[RCB22] Prasanna Ravi, Anupam Chattopadhyay, and Anubhab 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, 2022. https://eprint.iacr.org/2022/737. 107

[Res18] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018. 1

[RHHM17] Mélissa Rossi, Mike Hamburg, Michael Hutter, and Mark E Marson. A side-channel assisted cryptanalytic attack against QcBits. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 3–23. Springer, 2017. 80

[Riv92] Ronald L. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, April 1992. 10

[RL09] William Ryan and Shu Lin. *Channel codes: classical and modern*. Cambridge University Press, 2009. 7

[Rom92] Steven Roman. *Coding and information theory*, volume 134. Springer Science & Business Media, 1992. 24

[RSA78] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. 1

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1989. 18, 19

[Sen11] Nicolas Sendrier. Decoding one out of many. In *International Workshop on Post-Quantum Cryptography*, pages 51–67. Springer, 2011. 35, 83

[Sha48] Claude E Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948. 24

[Sha89] Adi Shamir. An efficient identification scheme based on permuted kernels. In *Conference on the Theory and Application of Cryptology*, pages 606–609. Springer, 1989. 4, 51, 54

[Sho94] Peter W Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE, 1994. 1

[SHR+22] Thomas Schamberger, Lukas Holzbaur, Julian Renner, Antonia Wachter-Zeh, and Georg Sigl. A power side-channel attack on the Reed-Muller Reed-Solomon version of the HQC cryptosystem. *Cryptology ePrint Archive*, 2022. 107

[Spe14] Joel Spencer. *Asymptopia*, volume 71. American Mathematical Soc., 2014. 70

[SRSWZ20] Thomas Schamberger, Julian Renner, Georg Sigl, and Antonia Wachter-Zeh. A power side-channel attack on the CCA2-secure HQC KEM. In *International Conference on Smart Card Research and Advanced Applications*, pages 119–134. Springer, 2020. 107

[SSPB19] Simona Samardjiska, Paolo Santini, Edoardo Persichetti, and Gustavo Banegas. A reaction attack against cryptosystems based on LRPC codes. In *International Conference on Cryptology and Information Security in Latin America*, pages 197–216. Springer, 2019. 80

[Ste88] Jacques Stern. A method for finding codewords of small weight. In *International Colloquium on Coding Theory and Applications*, pages 106–113. Springer, 1988. 35, 56, 60

[SV01] P Foggia C Sansone and M Vento. An improved algorithm for matching large graphs. In *Proc. of the 3rd IAPR-TC-15 International Workshop on Graph-based Representations*, 2001. 57

[SV19] Nicolas Sendrier and Valentin Vasseur. On the decoding failure rate of QC-MDPC bit-flipping decoders. In *International Conference on Post-Quantum Cryptography*, pages 404–416. Springer, 2019. 81, 103

[SV20a] Nicolas Sendrier and Valentin Vasseur. About low DFR for QC-MDPC decoding. In *PQCrypto 2020-Post-Quantum Cryptography 11th International Conference*, volume 12100, pages 20–34. Springer, 2020. 80, 84

[SV20b] Nicolas Sendrier and Valentin Vasseur. On the existence of weak keys for QC-MDPC decoding. Cryptology ePrint Archive, Report 2020/1232, 2020. https://ia.cr/2020/1232. 97, 105

[Til18] Jean-Pierre Tillich. The decoding failure probability of MDPC codes. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 941–945. IEEE, 2018. 80, 84

[TS16] Rodolfo Canto Torres and Nicolas Sendrier. Analysis of information set decoding for a sub-linear error weight. In *Post-Quantum Cryptography*, pages 144–161. Springer, 2016. 35, 83

[Ull76] Julian R Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1):31–42, 1976. 57

[Vas21] Valentin Vasseur. QC-MDPC codes DFR and the IND-CCA security of BIKE. Cryptology ePrint Archive, Report 2021/1458, 2021. https://ia.cr/2021/1458. xii, 80, 82, 83, 84, 85, 92, 97, 105

[vT93] Henk CA van Tilborg. Coding theory, a first course. *Lecture Notes on Error-Correcting Codes*, 3:11–13, 1993. 7, 24

[WR19] Matthew Walters and Sujoy Sinha Roy. Constant-time BCH error-correcting code. Cryptology ePrint Archive, Report 2019/155, 2019. https://eprint.iacr.org/2019/155. 49, 50

[WTBBG19] Guillaume Wafo-Tapa, Slim Bettaieb, Loic Bidoux, and Philippe Gaborit. A practicable timing attack against HQC and its countermeasure. Cryptology ePrint Archive, Report 2019/909, 2019. https://eprint.iacr.org/2019/909. 30, 49, 107

[YGH17] Yuval Yarom, Daniel Genkin, and Nadia Heninger. Cachebleed: a timing attack on openssl constant-time RSA. *Journal of Cryptographic Engineering*, 7(2):99–112, 2017. 21, 23

[Ylo96] Tatu Ylonen. SSH–secure login connections over the internet. In *Proceedings of the 6th USENIX Security Symposium*, volume 37, pages 40–52, 1996. 1