UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS REITOR JOÃO DAVID FERREIRA LIMA
UNDERGRADUATE PROGRAM IN COMPUTER SCIENCE

Filipe Oliveira de Borba

AN IND-CCA RANK METRIC ENCRYPTION SCHEME IMPLEMENTATION

Florianópolis, Santa Catarina – Brazil
2022

Filipe Oliveira de Borba

AN IND-CCA RANK METRIC ENCRYPTION SCHEME IMPLEMENTATION

Bachelor's Thesis submitted to the Undergraduate Program in Computer Science of Universidade Federal de Santa Catarina for degree acquirement in Bachelor of Science degree in Computer Science.
Supervisor: Ricardo Felipe Custódio, PhD.

Florianópolis, Santa Catarina – Brazil
2022

Legal Notes:

There is no warranty for any part of the documented software. The authors have taken care in the preparation of this thesis, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained here.

Cataloging at source by the University Library of the Federal University of Santa Catarina.
File compiled at 12:33h of the day Sunday 27th March, 2022.

Filipe Oliveira de Borba
An IND-CCA Rank Metric Encryption Scheme Implementation / Filipe Oliveira de Borba; Supervisor, Ricardo Felipe Custódio, PhD.; Co-supervisor, Daniel Panario, PhD. – Florianópolis, Santa Catarina – Brazil, 16 of March of 2022.
76 p.

Bachelor's Thesis – Universidade Federal de Santa Catarina, INE – Department of Informatics and Statistics, CTC – Technological Center, Undergraduate Program in Computer Science.

Includes references

1. Post-Quantum Cryptography, 2. Code-Based Cryptography, 3. Rank Metric, 4. PKE, 5. IND-CCA, I. Ricardo Felipe Custódio, PhD. II. Daniel Panario, PhD. III. Undergraduate Program in Computer Science IV. An IND-CCA Rank Metric Encryption Scheme Implementation

CDU 02:141:005.7

Filipe Oliveira de Borba

AN IND-CCA RANK METRIC ENCRYPTION SCHEME IMPLEMENTATION

This Bachelor's Thesis was considered appropriate to get the Bachelor of Science degree in Computer Science, and it was approved by the Undergraduate Program in Computer Science of INE – Department of Informatics and Statistics, CTC – Technological Center of Universidade Federal de Santa Catarina.

Florianópolis, Santa Catarina – Brazil, 16 of March of 2022.

Jean Everson Martina, PhD.
Coordinator of Undergraduate Program in Computer Science

Examination Board:

Ricardo Felipe Custódio, PhD.
Supervisor
Universidade Federal de Santa Catarina – UFSC

Daniel Panario, PhD.
Co-supervisor
Carleton University – CU

Florian Reneld Ghislain Caullery, PhD.
Qualcomm Technologies Incorporated – QTI

Thaís Bardini Idalino, PhD.
Universidade Federal de Santa Catarina – UFSC

ACKNOWLEDGEMENTS

Gostaria de agradecer à minha mãe, ao meu irmão, às minhas avós, e minha namorada, pelo seu carinho, apoio e torcida. Agradecimentos especiais aos meus orientadores, Custódio e Daniel, por todo o suporte e confiança, mesmo nos momentos mais difíceis, e ao Florian pelo convite para a pesquisa que deu início a este trabalho. Agradeço também meu amigo Eduardo, que despertou em mim o desejo da pesquisa.

ABSTRACT

The advances in the field of quantum computation impose a severe threat to the cryptographic primitives used nowadays. In particular, the community predicts public-key cryptography will be turned completely obsolete if these computers are ever produced. In the light of these facts, researchers are contributing in a great effort to preserve current information systems against quantum attacks. Post-quantum cryptography is the area of research that aims to develop cryptographic systems to resist against both quantum and classical computers while assuring interoperability with existing networks and protocols. This work considers the use of Gabidulin codes—a class of error-correcting codes using rank metric—in the construction of encryption schemes. We first introduce error-correcting codes in general and Gabidulin codes in particular. Then, we present the use of these codes in the context of public-key encryption schemes and show that, while providing the possibility of smaller key sizes, they are especially challenging in terms of security. We present the scheme proposed in Loidreau in 2017, showing that although correcting the main weakness in previous propositions, it is still insecure related to chosen-ciphertext attacks. Then, we present a modification to the scheme, proposed by Shehhi et al. to achieve CCA security, and provide an implementation. We also analyze the theoretical complexity of recent attacks to rank-based cryptography and propose a set of parameters for the scheme.

Keywords: Post-Quantum Cryptography. Code-Based Cryptography. Rank Metric. PKE. IND-CCA.

LIST OF FIGURES

LIST OF FRAMES

LIST OF TABLES

LISTINGS

LIST OF ABBREVIATIONS AND ACRONYMS

CCA	Chosen-Ciphertext Attack
CPA	Chosen-Plaintext Attack
KEM	Key Encapsulation Mechanism
NIST	National Institute of Standards and Technology
PKE	Public-Key Encryption
XOF	Extendable-Output Function

CONTENTS

Part I

Research

1 INTRODUCTION

Our world runs on software. From sensor networks to online banking, from social networks to electronic voting. These applications shape our economy, society, and the way we live like never seen before. Nonetheless, to be useful, these systems need to exchange information. More important, this communication must happen in a manner that unauthorized parts do not participate in the sense that information transmitted is not disclosed to them for as long as its secrecy is necessary.

Cryptography provides the basic building blocks which cryptographic functionalities develop on top to secure such systems. It divides into two broad classes of methods to implement before-mentioned functionalities: asymmetric cryptography, also known as public-key cryptography, which makes use of a pair of distinct but related keys to achieve its goal, and symmetric cryptography, that makes use of a unique key for such. Then, the keys are pieces of information used as inputs for these functions so that the output of operations using one of the keys can only be reverted using the other one. Here we shall only discuss the former.

Three of the main cryptographic functionalities used nowadays are public-key encryption, digital signatures, and key exchange, and many of the most crucial communication protocols rely on them. These functionalities are currently implemented primarily using Diffie-Hellman key exchange, the RSA cryptosystem, and elliptic curve cryptosystems, all of which belong to the class of public-key cryptography methods. They rely on well-known number theoretical problems such as the integer factorization and the discrete log problem to create one-way functions.

It happens that while these problems are considered to have no efficient solution on a classical computer, it turns out not to be true for quantum computers—computers that make use of the properties of quantum mechanics to perform computations. Although it is not yet clear what class of problems a quantum computer can solve, strong evidence suggests it has computational powers exceeding those of classical computers. Remarkable results in the field due to Shor (1997) and Grover (1996) give further evidence to this belief. While Grover's algorithm does not turn current cryptographic technologies obsolete, it offers quadratic speedup for problems relating to searching, collision finding, and the evaluation of Boolean formulae, thus imposing a severe threat to symmetric-key cryptography. On the other hand, Shor's discovery renders public-key cryptography completely useless by solving both problems mentioned earlier, which all public-key cryptography relies upon, in subexponential time using a quantum computer.

The need for a new class of algorithms that can offer resistance to quantum computers becomes more urgent as the field of quantum computation evolves. Since Shor, the theory of quantum algorithms has developed significantly and is unpredictable how advantageous can be the use of quantum mechanics for information processing.

Another concern is that of when will quantum computers be built in large-scale. While this question was still unclear at the time when Shor and Grover first published its findings, it does not seem to be the case now, and many scientists expect quantum computers to be widely available in the next ten or twenty years. For them, the physical implementation of these machines is simply a question of engineering.

Taking into account these facts, an international community composed by academia, industry, and government organizations, is working on the task of developing, testing, and standardizing new quantum-resistant primitives. To this new field of study is given the name of post-quantum cryptography in allusion to the scenery after the deployment of quantum computers, although for the quantum skeptic, this is a misnomer to the quest. Among these initiatives, the most prominent is that of the National Institute of Standards and Technology of the United States. The agency initiated a process to standardize post-quantum cryptographic primitives that offer resistance to both classical and quantum computers. Besides that, these primitives should interoperate with current systems and protocols. The role of NIST in the standardization of cryptography is well-known due to its Advanced Encryption Standard competition that selected Rijndael as the encryption algorithm to be used by the U.S Government and, voluntarily, by the private sector.

As mentioned before, the impact of these new technologies on symmetric cryptography is not as severe as it is for the public-key counterparts. In particular, Grover's algorithm provides only a quadratic improvement on current algorithms as opposed to Shor's, which provide exponential improvements. Furthermore, it has been shown that exponential speedups for search algorithms are not feasible and, therefore, symmetric algorithms and hash functions may be useful in a quantum world. Given that, the focus is on public-key algorithms on the NIST standardization process.

The institute expects to standardize at least one proposal for each public-key encryption, digital signatures, and key exchange protocols. However, in contrast to the previous processes that led to the Advanced Encryption Standard and the Secure Hash Algorithm-3, this time the process is not in a competition format. Instead, NIST sees it as an opportunity of achieving community consensus in a transparent and timely manner. As a result, more than one proposal in each category can be seen as good choices and recommended by the institute. Currently, the process is in its third round. Among the main families of proposed post-quantum primitives are hash based signatures, lattice-based cryptography, multivariate polynomial cryptography, and code-based cryptography.

This work concerns a specific branch of the latter which relies on rank metric codes. The objective is to introduce post-quantum cryptography using rank metric codes through the example of a cryptosystem, proposed by Shehhi et al. (2019), and an implementation of it. Along the way we give an overview of this field of research, showing the origins of rank metric cryptography with the work in Gabidulin, Paramonov,

and Tretjakov (1991), the cat-and-mouse game played by variants of this scheme and attacks breaking them, and a proposal that rules out polynomial-time attacks, and discuss how recent attacks impacts on the key sizes for the scheme we implement.

Chapter 2 provides a brief introduction to the topics necessary to understand the rest of the work. We start by defining a public key encryption scheme and show how one can prove a scheme secure from security assumptions and definitions. Then, it introduces finite fields and the most relevant properties to understand the rest of the work. The chapter finishes with a gentle introduction to coding theory and presents Gabidulin codes with examples.

In Chapter 3, we present cryptography based on Gabidulin codes, starting with the GPT cryptosystem deriving McElice's to rank metric and the structure of Gabidulin codes allowed a myriad of attacks, including the Overbeck attack that has polynomial time complexity. We show many attempts to patch the scheme failing. Then, we present a proposal by Loidreau that effectively defends against the Overbeck attack and its CCA-secure variant proposed by Shehhi et al. We conclude by demonstrating through experiments how recent attacks have decreased the security of these systems, forcing them to use larger keys and select the parameters used in our implementation based on the results.

Finally, Chapter 4 presents and explains in detail our implementation of Shehhi et al.: its dependencies and how keys are structured and encoded into bytes so that one can import and export them. Then we map each line of the code for the three algorithms composing the cryptosystem to the corresponding steps in their definitions. In the end, we demonstrate how to use the implementation through an example of communication between Alice and Bob.

Chapter 5 concludes the thesis.

2 BACKGROUND

2.1 PUBLIC-KEY ENCRYPTION

As mentioned in the Chapter 1, in private-key encryption the key used to encrypt a message is the same that should be used to decrypt this very message, or at least the keys are related in a manner that is very easy to compute one key from another and we can assume they are the same for simplicity. This means that if Alice and Bob want to communicate, they must agree on the keys to use and keep them secret. Moreover, if they want to communicate with any other party, they need to agree on new secret keys with these parties. If Alice, for example use the same key shared with Bob to communicate with Charles, then the latter can use the key to eavesdrop the communication between Alice and Bob.

The public-key revolution led by (Diffie and Hellman, 1976) allowed private communication without the previous establishment of secret keys. Public-key encryption makes use of a pair of distinct but related keys, the public key and the private key. Contrary to the private-key counterpart, only the private key can derive the public one and a message encrypted with the latter can only be decrypted with the former. This allows Alice to sent Bob her public key in the clear without risking someone to read the messages she receives. There are still issues like how Bob knows a public key is in fact Alice's public key and how Alice can be sure that the message she receives is from Bob. These issues are treated by other parts of cryptography and we won't touch the subject here.

We now define, in a generic manner, a public-key encryption scheme and then discuss some characteristics relevant to our discussion.

Definition 2.1 (Public-Key Encryption Scheme)**.** A public-key encryption scheme is a triple of probabilistic polynomial time algorithms $\Pi = (\text{GEN}, \text{ENC}, \text{DEC})$, where:

- The key generation algorithm GEN takes as the security parameter N and outputs a pair of keys (pk, sk), where pk is the public key and sk is the private one.
- The encryption algorithm ENC takes a public key pk and a message m as input and outputs a ciphertext c.
- The decryption algorithm DEC takes as input the secret key sk and a ciphertext c and outputs either a message m corresponding to the input ciphertext or a failure symbol we denote by $\perp$.

2.2 SECURITY PROOFS

Cryptography is usually referred to as *the art of hiding information*. And, in fact, for a long time hiding information was an exercise of inventiveness, with the design and analysis of encryption schemes done in an ad hoc fashion. The general workflow used to be the identification of a flaw in such schemes, followed by a patch to hinder the attack. No definition of what requirements an encryption scheme should satisfy to be considered secure and no way to prove the security of such schemes existed. Nowadays, cryptography evolved into more of a science, relying on three principles to prove the security of a system: definitions, assumptions, and proofs.

2.2.1 Security Definitions

Security definitions state precisely the security guarantees a scheme provides and what threats are present in the environment. The security guarantees refer to results the system should prevent attackers from achieving and the threat model to what capabilities the attackers have.

Definitions support designers of cryptographic primitives to make the best possible use of computational resources while still confident it attains the required level of security. On the other hand, a scheme cannot be claimed insecure without these definitions in place. Consider the algorithms post-quantum cryptography wants to replace. They are not secure against quantum computers. However, if the cost of using a quantum computer to obtain a piece of information surpasses the return, an attacker is unlikely to spend resources on this purpose. Then, one can define a system that does not regard this kind of attack and use algorithms that might be more efficient or have shorter keys. Having formal definitions also helps users evaluate which primitives are suitable for an application and choose which one fits better. But the benefit that shines brighter for them is the possibility of substituting one primitive with another. Users can replace a primitive with another because it is more efficient, or perhaps the old one has been broken because one of its assumptions no longer holds. We'll talk about assumptions in the next section.

Although it seems straightforward to formulate an informal notion of what it means for a scheme to be secure, we can argue to the contrary. Let's start with Kerckhoff's principle that states the key is the only component of a cryptographic primitive that should be kept secret and define the encryption scheme not to leak the key, i.e., the attacker should not recover the key. Now, consider an encryption algorithm in which the the encryption of a message is the message itself. It leaks no information about the key since the ciphertext does not depend on the key at all, and nevertheless, it is far from secure. We might strengthen our requirements and aim for a guarantee that the attacker cannot recover the message. After all, the goal is to protect the communication, not the key. However, this definition is not enough to claim a scheme secure; no bank

would like to have data of half of its clients exposed, not even the number of digits in their current accounts. It is clear from this observation that a more restrictive definition is necessary.

There are many subtleties in defining what it means for an encryption scheme to be secure. In particular, the point stressed in the last paragraph needs to be formally defined. If the attacker knows the message sent is an email, she knows some characters present in the header. Moreover, she may know in advance or even guess other information in the email, such as the email domain, sender, recipient, and contents. For this reason, the security guarantee must take into account the prior knowledge and or the expectations the attacker has about the contents of the encrypted messages. A better definition would ask that regardless of any prior knowledge an attacker has, a ciphertext must not give any additional information about the underlying plaintext. Here, we assume that multiple messages are encrypted using the same key, so the resulting ciphertexts must not leak information on any of the plaintexts. We give a precise definition of a security guarantee after introducing indistinguishability experiments.

As explained, the threat model specifies the types of attackers the scheme should defend against by defining their capabilities. Consequently, if an attacker can take some action not stated in the model, the security guarantees do not necessarily hold for this attacker. In this case, the model does not capture all of the attacker's abilities, or perhaps it considers that such an attack is unlikely, as in the case we mentioned before. Now we present the two most widely used threat models. Alongside, we introduce experiments that help to define secure encryption against the modeled adversaries.

2.2.1.1 Chosen-Plaintext Attack

The first model is the chosen-plaintext attack (CPA), presented in the context of the experiment in Definition 2.2. In this model, the attacker has the capability of obtaining ciphertexts corresponding to plaintexts of its choice. We represent this ability by giving the attacker access to an encryption oracle that, given a message m, returns the encryption of m using the public key pk. In the context of public-key encryption, this is the minimum required of a threat model because, as pointed in Section 2.1, the key used is assumed to be publicly available, and the attacker is free to use it according to its will.

Definition 2.2 (The CPA indistinguishability experiment)**.** Let $\Pi = (\textsc{Gen}, \textsc{Enc}, \textsc{Dec})$ be a public-key encryption scheme and let A be an adversary composed of a pair of probabilistic polynomial time algorithms. We define the chosen-plaintext attack indistinguishability experiment $\textsc{Exp}^{\mathsf{cpa}}_{\mathrm{A},\Pi}(1^n)$ as the following sequence of steps:

1. A key pair $(pk, sk) = \textsc{Gen}(1^n)$ is generated.

2. A is given input 1^n and oracle access to $\textsc{Enc}_{pk}(\cdot)$. It outputs a pair of messages (m_0, m_1) of same length.
3. A uniform bit b is chosen.
4. A ciphertext $c \leftarrow \textsc{Enc}_{pk}(m_b)$ is computed. We call c the challenge ciphertext.
5. A tries to figure out the value b and outputs a bit b' corresponding to its answer.
6. Return 1 if $b' = b$ and 0 otherwise.

With this definition in hands, we can define precisely the security guarantee we only stated informally before. Definition 2.3 takes its randomness from the attacker A, the encryption scheme Π, and the sampling of b in Step 3 of the CPA indistinguishability experiment. One can interpret it as follows. Take the probabilities of the adversary succeeding and failing. If they are the same, i.e., 1/2, one can say that the adversary cannot distinguish if c is the encryption of m_0 or m_1 since these are the probabilities one would get for an attacker that only makes random guesses. However, we allow the attacker to succeed with a chance a little better than guessing, more precisely $1/2 + \epsilon$.

Definition 2.3 (Indistinguishability under chosen-plaintext attacks)**.** Let Π be a public-key encryption scheme. Then, Π has indistinguishable encryptions under a chosen-plaintext attack, or is CPA-secure, or is IND-CPA, if for all adversaries A, there is a negligible function ϵ such that

$$\Pr\left(\textsc{Exp}^{\mathsf{cpa}}_{\mathrm{A},\Pi}(1^n) = 1\right) \leq \frac{1}{2} + \epsilon(1^n).$$

This extra chance is necessary to model situations in which, instead of guessing which message was encrypted, the adversary obtains, maybe in a randomized manner, some information that helps to find out the correct answer. Consider the situation in which A requests the oracle to encrypt messages $m_2, m_3, \ldots, m_\ell$, obtaining ciphertexts $c_2, c_3, \ldots, c_\ell$. Then, independently of the choice of m_0 and m_1, it can guess a random private key k and check whether $\textsc{Dec}_k(c_i) = m_i$ for $2 \leq i \leq \ell$, and in the affirmative case, use k to decrypt c, succeeding in the experiment. Then, the extra chance $\epsilon(1^n)$ of succeeding is exactly $(\ell - 1)/|K|$, where K is the key space, and, if $|K|$ is exponential in n, it is negligible for any polynomial-time adversary.

Of course, this is not the only way to recover the key, and more elaborate attacks may exist. The definition of a chosen-plaintext attack left this avenue open, describing only the tools the attacker has under his belt to perform his strategy.

2.2.1.2 Chosen-Ciphertext Attack

The other threat model is the chosen-ciphertext attack (CCA), in which the adversary, in addition to the previous model capabilities, can obtain decryptions for cipher-

texts of its choice. Definition 2.4 introduces the chosen-ciphertext attack indistinguishability experiment.

Definition 2.4 (The CCA indistinguishability experiment)**.** Let $\Pi = (\text{GEN}, \text{ENC}, \text{DEC})$ be a public-key encryption scheme and let $\mathrm{A} = (\mathrm{A}_1, \mathrm{A}_2)$ be an adversary composed of a pair of probabilistic polynomial time algorithms. We define the CCA indistinguishability experiment $\text{EXP}^{\text{cca}}_{\mathrm{A},\Pi}(1^n)$ as the following sequence of steps:

1. A key pair $(pk, sk) \leftarrow \text{GEN}(1^n)$ is generated.
2. A_1 is given input 1^n and oracle access to $\text{ENC}_{pk}(\cdot)$. It outputs a pair of messages (m_0, m_1) of same length.
3. A uniform bit b is chosen.
4. A ciphertext $c \leftarrow \text{ENC}_{pk}(m_b)$ is computed. We call c the challenge ciphertext.
5. A_2 is given input 1^n and c, and oracle access to $\text{ENC}_{pk}(\cdot)$. Then, A_2 tries to figure out the value b and outputs a bit b' corresponding to its answer.
6. Return 1 if $b' = b$ and 0 otherwise.

This experiment is identical to the one in Definition 2.2, except for the decryption oracle access in Steps 2 and 5, and one additional requirement: after receiving the challenge ciphertext in Step 4, the adversary cannot query the encryption of this very same ciphertext. Otherwise, he will obtain the message m_0 or m_1 which generated it, assuming perfect correctness of the scheme.

The definition of a chosen-ciphertext attack seems too strong to concern about in practice. However, padding-oracle attacks work using the definition of a chosen-ciphertext attack to some degree; instead of requiring the corresponding plaintext, the attacker only needs to know if the ciphertext decrypted correctly. For example, a server might request re-transmission or even close connection if the decryption of a ciphertext is not a valid message. Next, we define what is necessary to consider a scheme secure under a chosen-ciphertext attack.

Definition 2.5 (Indistinguishability under chosen-ciphertext attacks)**.** Let Π be a public-key encryption scheme. Then, Π has indistinguishable encryptions under a chosen-ciphertext attack, or is CCA-secure, or is IND-CCA, if for all probabilistic adversaries A, there is a negligible function ϵ such that

$$\Pr\left(\text{EXP}^{\text{cca}}_{\mathrm{A},\Pi}(1^n) = 1\right) \leq \frac{1}{2} + \epsilon(1^n).$$

Again, the definition of a chosen-plaintext attack does not specify how an attacker may work to achieve its objective. This fact is stated in the requirement that the inequality

holds for all probabilistic polynomial-time adversaries. Notice also that the extra chance epsilon expresses the possibility the adversary succeeds in some experiments as long as this number is negligible compared to the number of failures. Next, we turn our focus into a crucial part of public-key cryptography: assumptions.

2.2.2 Assumptions

The security of a cryptosystem often relies on assumptions that some problems are difficult to solve computationally or that some construction used as building blocks satisfies some properties. For the first one, the theory of computational complexity provides us with well-studied problems that, up to these days, still don't have an efficient solver—at least without a quantum computer. Examples include the planted clique, the random SAT, the discrete logarithm, and the integer factoring. Therefore, for schemes relying on such problems, breaking means solving a long-standing open problem in computer science, and any other construction relying on it is also insecure and needs at least a redesign.

Concerning building blocks used in cryptography, things are a bit different. Again, breaking them means breaking the scheme. However, in this case, substituting one building block with any other that does satisfy the required properties may fix the cryptosystem. An example is a pseudo-random number generator used in all cryptosystems or, as we will see later in our implementation, a hash function. Breaking such a function means no scheme shall rely on it. However, there is no need to redesign the scheme. It suffices to replace the hash function with another attending the requirements to reestablish security.

2.2.3 Proofs

Proofs are the glue that binds things together to claim a cryptosystem is secure. A proof may assume some facts and show a scheme achieves a precise security guarantee against a specific type of attacker. As an example, we may want to prove a scheme CCA-secure. So, we have defined the attackers the scheme needs to defend against and what it means to resist successfully. We may start from Definition 2.4 and Definition 2.5 that give a framework that deals exactly with the attackers and guarantees we want. If our cryptosystem relies on some building block or computational problem, we may add these as assumptions. Then, we manipulate the experiment in Definition 2.4, trying to show that the inequality in Definition 2.5 holds, and if so, our proof is complete.

It is important to stress that any modification in the elements used in a proof nullifies its validity. A misconception often occurs when dealing with quantum computers. Since we speculate that these computers can efficiently solve problems that classical computers cannot, people claim that the proofs are probably wrong and the related cryptosystem is not secure. Even if quantum computers break the adjacent scheme

eventually, what happens is that one of the assumptions used in the proof no longer holds and is not a mathematical or logical error.

2.3 FINITE FIELDS

The theory of finite fields is a branch of abstract algebra emerging in the last century to the general interest due to its applications to coding theory and cryptography, both discussed here, and to other topics such as combinatorics and the study of switching circuits. The subject has its origins with prominent mathematicians such as Pierre de Fermat, Leonhard Euler, Joseph-Louis Lagrange, and Adrien-Marie Legendre. The contribution from these names to the theory was the so-called finite prime fields—finite fields with a prime number of elements. A more general theory of finite fields is said to have started with the work of Carl Friedrich Gauss and Évariste Galois. In this section, we only touch the surface of this rich subject, covering what is necessary for the rest of the work. For a thorough introduction, we refer the interested reader to Lidl and Niederreiter (1996) and for an extensive collection of results in finite fields, see Mullen and Panario (2013).

2.3.1 Definition and Notation

Definition 2.6 (Group)**.** A group is a non-empty set G equipped a binary operation denoted $*$, closed on G, such that the following properties hold:

1. Associativity: $(a * b) * c = a * (b * c)$, for all $a, b, c \in G$.
2. Existence of an identity or unit element: there is an element $e \in G$, such that $a * e = e * a = a$, for all $a \in G$.
3. Existence of an inverse for every element: for each $a \in G$, there exists an inverse element a^{-1}, such that $a * a^{-1} = a^{-1} * a = e$.

If in addition it holds that $a * b = b * a$, for all $a, b \in G$, the group is said to be commutative or Abelian. From the definition of a group, it is easily noted that the identity element e and the inverse element a^{-1} of any $a \in G$ are uniquely determined. In discussing groups, we shall often use either the multiplicative or the additive notation. $+$ and $\cdot$ respectively, for the binary operator although these operations does not necessarily have the usual semantics. For the first, we denote $a * b$ as $a \cdot b$ or simply as ab and the inverse element as a^{-1}. Furthermore, we denote the n-fold composition of $a \in G$ with itself, for $n \in \mathbb{N}$, as

$$a^n = \underbrace{aa \ldots a}_{n \text{ times}}.$$

The later notation is usually reserved to refer to a commutative group, in which case we denote the operator as $+$, the inverse element of $a \in G$ as $-a$, and the n-fold composition as

$$na = \underbrace{a + a + \cdots + a}_{n \text{ times}}.$$

Definition 2.7 (Field)**.** A field is a non-empty set F equipped with two binary operations, $+$ and $\cdot$, such that:

1. F together with $+$ is a commutative group.
2. The set $F^{\times} = F \setminus \{0\}$, where 0 is the identity of $+$, forms a commutative group with respect to $\cdot$.
3. Left distributivity: $a \cdot (b + c) = a \cdot b + a \cdot c$, for all $a, b, c \in F$.
4. Right distributivity: $(b + c) \cdot a = b \cdot a + c \cdot a$, for all $a, b, c \in F$.

Definition 2.8 (Finite field)**.** A finite field, denoted $\mathbb{F}$, is a field with a finite number of distinct elements.

We shall not enter the theory behind the construction of finite fields as it is not necessary to the topic. For the purpose of this work it suffices to say that finite fields have either p or p^n elements, where p is a prime and n is a positive integer. p is the characteristic of the finite field and the number of elements it contains is called its order. Any two finite fields of same order are equivalent up to isomorphism.

The finite field with p elements is the set $\{0, 1, \ldots, p-1\}$ together with arithmentic modulo p. These are the prime finite fields mentioned earlier and we denote them as $\mathbb{F}_p$. A very useful example is the binary field $\mathbb{F}_2$. The tables below, referred to as Cayley tables, show how addition and multiplication works on the binary field $\mathbb{F}_2$

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

Table 1 – Cayley tables for addition and multiplication in $\mathbb{F}_2$

For the latter case, a finite field with p^n elements that we denote $\mathbb{F}_{p^n}$ is formed by the polynomials of degree less than n with coefficients in $\mathbb{F}_p$, and the operations of addition and multiplication are performed modulo an irreducible polynomial of degree n, again over $\mathbb{F}_p$. An irreducible polynomial over a finite field is equivalent to a prime number over the natural numbers and cannot be factored into two polynomials greater than 1. As an example we show the Cayley tables for the finite field $\mathbb{F}_{2^2} \cong \mathbb{F}_4$, with operations modulo $m(x) = x^2 + x + 1$.

$+$	0	1	x	$x+1$
0	0	1	x	$x+1$
1	1	0	$x+1$	x
x	x	$x+1$	0	1
$x+1$	$x+1$	x	1	0

Table 2 – Cayley table for addition in $\mathbb{F}_4$

$\cdot$	0	1	x	$x+1$
0	0	0	0	0
1	0	1	x	$x+1$
x	0	x	$x+1$	1
$x+1$	0	$x+1$	1	x

Table 3 – Cayley table for multiplication in $\mathbb{F}_4$

An often useful representation of a finite field of order p^n, especially in coding theory, is to view the coefficients of the polynomials in $\mathbb{F}_{2^n}$ as a coordinate of a vector of length n. So, in the previous example, we write the polynomial $p(x) = x$ as the vector $\begin{pmatrix}1 & 0\end{pmatrix}$. In this case, the finite field is a vector space over $\mathbb{F}_p$ and we denote it as $\mathbb{F}_p^n$.

2.3.2 Properties

In this section, we define a few properties concerning finite fields as well as polynomials and matrices over them.

Lemma 2.1. Let F be a finite field with q^n elements. Then, $\alpha^{q^i} = \alpha^{q^{(i \mod n)}}$ for any $\alpha \neq 0 \in F$ and any integer i.

The previous lemma implies that $\alpha^{q^n-1} = 1$ and the inverse of α is α^{q^n-2}, since we have $\alpha^{q^n-2} \cdot \alpha = \alpha^{q^n-1} = 1$.

Definition 2.9 (Linearized polynomial)**.** A polynomial $p(x)$ is a linearized polynomial if it has the form

$$p(x) = \sum_{i=0}^{d} a_i x^{q^i}, \quad a_i \in \mathbb{F}_{q^m},\ \forall i \in [0, d]. \tag{1}$$

Definition 2.10 (Subfield mapping)**.** Let $B = \beta_1, \ldots, \beta_m$ be a basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. The subfield mapping of a vector $\mathbf{v} = \begin{pmatrix}\alpha_1 & \ldots & \alpha_n\end{pmatrix}$ in $\mathbb{F}_{q^m}^n$ with respect to B is given by the map $f_B(\mathbf{v}) = (a_{ij}) \in \mathbb{F}_q^{m \times n}$ such that

$$\alpha_i = \sum_{j=1}^{m} a_{ij}\beta_j, \quad 1 \leq i \leq n.$$

Example 2.3.1. As an example, let $\beta_1 = x^3 + x + 1$ and $\beta_2 = x^3 + x^2 + 1$ form a basis B of $\mathbb{F}_{2^4}$ over $\mathbb{F}_{2^2}$. Then, the subfield mapping of $\mathbf{v} = \begin{pmatrix} 1 & 0 & 1 & 0 \end{pmatrix}$ with respect to B is

$$f_B(\alpha) = \begin{pmatrix} x+1 \\ x+1 \end{pmatrix}.$$

It is easy to verify that $f_B(\alpha)$ satisfies the definition:

$$\begin{aligned} a_{11}\beta_1 + a_{21}\beta_2 &= (x+1)(x^3+x+1) + (x+1)(x^3+x^2+1) \\ &= (x^3 + x^2 + x) + x^2 \\ &= x^3 + x \cong \alpha. \end{aligned}$$

2.4 CODING THEORY: AN INTRODUCTION

The field of coding theory has its origins in the seminal publication by Claude Shannon in 1948, where the author guarantees the existence of codes that can transmit information at rates close to the channel's capacity with an arbitrarily small probability of error. Since then, algebraic coding theory has evolved, developing tools to construct codes that can achieve these rates.

Algebraic coding theory is used to transmit information in a plethora of ways: digital television, bar code scanners, radio waves, and magnetic disks, just to cite a few. However the channel of communication used to transmit and the means used to store information are not ideal in the sense the the information may may be distorted. The problem algebraic coding theory aims to solve is the correct storage and delivery of information through a noisy channel, where the "noise" we refer to may be human error, thermal noise, physical deterioration, etc.

2.4.1 A first example

As an introductory example, consider we want to send a message composed by a sequence of 0s and 1s of length 1024. Assume further that errors occur independently, with probability 0.01. Thus, the probability $P(X)$ of receiving an error-free message is

$$P(X) = 0.99^{1024} \approx 0.00003.$$

On the other hand, if we use a simple coding scheme, the threefold repetition scheme, which consists of sending each digit three times on transmission and decoding each block of three digits using the majority rule, we can achieve a much better result. Again, if we consider the occurrence of errors is independent, the probability of an error to occur in any block, say a block encoding a 0, is then the probability of receiving one

of 111, 110, 101, 011. That is, the probability of errors occurring in all three position plus three times the probability of errors occurring in two positions

$$\begin{aligned} P(X) &= (0.01)(0.01)(0.01) + 3 * (0.01)(0.01)(0.99) \\ &= 0.000001 + 3 * 0.000099 \\ &= 0.000298. \end{aligned}$$

Therefore, the probability of receiving an error-free message is greater than 0.9997. Compare this with the result of sending a message without any encoding; it justifies the study and use of algebraic coding theory to transmit information over noisy channels. Despite the better results in transmission of information of the threefold coding scheme, this class of codes, called repetition codes, are far from efficient. For this one specifically, two thirds of all data transmitted is useless in the sense that it does not contain relevant information.

From the example, we see that the objective of algebraic coding theory is to devise methods to encode and decode information in a reliable and efficient manner. We have an encoding method where we add redundancy in a systematic manner and a decoding method where we use this redundancy to detect and occasionally correct errors in transmission.

2.4.2 A classical example: the $(7, 4)$ Hamming code

Now we turn our attention to a more elaborated and the most widely used class of codes: Hamming codes, named after Richard Hamming in 1948.

The example we show here is a $(7, 4)$ Hamming code. This notation means a message of four symbols long is encoded with seven symbols. That is, we add three symbols of redundancy. Again, consider out alphabet to be the set $\{0, 1\}$. This simple encoding scheme can be suitably depicted by a Venn diagram as shown in Figure 1. Start by placing the symbols in the message in regions 1, 2, 3, and 4. Then, for each region I, II, and III, put 0 on it if the correspondent circle has an even number of 1s. Otherwise, put 1.

To check a received code word for correctness, it suffices for us to place the symbols received in regions 1, 2, 3, 4, I, II, III, in order, and then analyze the Venn diagram. For example, consider the Venn diagram for the received code word 0011010 in Figure 2. In this example, C has the correct parity; the number of 1s is even. But this is not the case, for neither A or B. Therefore the error is in the intersection of these two circles, i.e., in region 1.

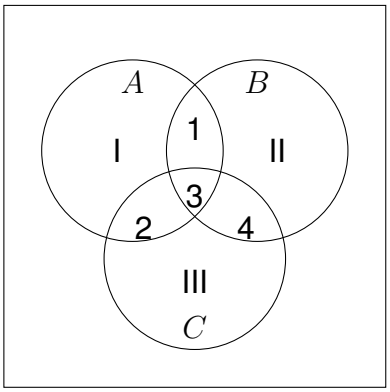


Figure 1 – Venn diagram for code construction

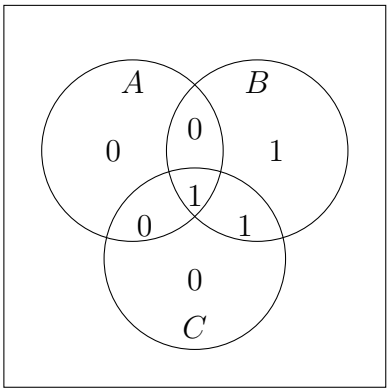


Figure 2 – Venn diagram for received word 0011010

2.4.3 Generator and Parity-check matrices

The example above shows a graphic manner to detect and to correct errors in a $(7,\ 4)$ Hamming code. But the Venn diagram does not tell how to encode a message and it is not suitable to any code. In special if the code is not from the family of Hamming codes. Moreover, there no way of deriving and proving properties using it.

For the purpose of representing a code, one of the two matrices we present in this section are normally employed. They are used interchangeably since they are algebraically related to one another. But first we define formally a linear code, which is the type of code we refer to in this work.

Definition 2.11 (Linear code)**.** An (n, k) linear code $\mathcal{C}$ over a finite field $\mathbb{F}_q$ is a k-dimensional subspace of $\mathbb{F}_q^n$.

That is, the encoding scheme represents messages as elements in the vector space $\mathbb{F}_q^k$ and encodes them by applying a linear transformation from $\mathbb{F}_q^k$ into $\mathbb{F}_q^n$. Such a transformation carrying elements from on vector space into another is performed by a $k \times n$ full-rank matrix $\mathbf{G}$. It is called the generator matrix of the code since it effectively generates the code from the message space. Take into consideration the code in example of Section 2.4.2. A possible generator matrix for the $(7, 4)$ Hamming

code on it is

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}. \tag{2}$$

The generator matrix for a given code is not unique as any $\mathbb{F}_q$-linear combination of its rows also generates the same code. On the decoding side is the second matrix used to represent a code, called the parity-check matrix and usually denote by $\mathbf{H}$. A parity-check matrix of a code relates to the corresponding generator by the equation $\mathbf{G}\mathbf{H}^\top = \mathbf{0}$. For the generator $\mathbf{G}$ in Equation 2 we can obtain the following parity-check matrix

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}. \tag{3}$$

The matrix receives this name because one of its roles is to detect occurred errors, which is done through the calculation of a vector called syndrome. A syndrome is always related to a received word, be it a valid codeword or a codeword distorted by errors. Then, we say that $\mathbf{s}$ is the syndrome of $\mathbf{c}$ if $\mathbf{s} = \mathbf{c}\mathbf{H}^\top$. For a codeword transmitted without errors its syndrome should be $\mathbf{0}$ and can be justified by observing that for any $\mathbf{m}$ in $\mathbb{F}_2^k$, we have $\mathbf{s} = \mathbf{m}\mathbf{G}\mathbf{H}^\top = \mathbf{0}$.

To finish this section we show a bit of how these matrices work. Suppose we want to encode a message $\mathbf{m} = \begin{pmatrix} 1 & 0 & 0 & 1 \end{pmatrix}$. To that end, it suffices to multiply $\mathbf{m}$ to the right by $\mathbf{G}$, which in this case produces $\mathbf{c} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$ as codeword. Then, if we calculate the syndrome of $\mathbf{c}$, the result is $\mathbf{0}$. Now, if a received word contain errors, say $\mathbf{c}' = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$ as in Section 2.4.2, the calculated syndrome is $\mathbf{s}' = \begin{pmatrix} 1 & 1 & 0 \end{pmatrix}$, indicating an error occurred.

From this example, we can define informally two important notions concerning error-correcting codes. The first of them is the notion of Hamming weight which, for any vector $\mathbf{v}$, is defined as the number of coordinates of V that differs 0. So, the Hamming weight of $\mathbf{c}'$ and $\mathbf{s}'$ are 3 and 2, respectively. The second concept is the Hamming distance between vectors, defined, for any vectors $\mathbf{u}$ and $\mathbf{v}$, as the Hamming weight of $\mathbf{u} - \mathbf{v}$ or, equivalently, the number of coordinates in which $\mathbf{u}$ diverges from $\mathbf{v}$.

In the next section, we define notions of weight and distance with greater detail and rigor when we introduce a family of codes using a different measure of distance.

2.5 GABIDULIN CODES

Gabidulin codes are an especial class of error-correcting codes using rank distance instead of the more widely known Hamming distance used in the example of

Subsection 2.4.2. These codes were first introduced in Delsarte (1978), where the author presents the concept of a rank metric code, proves an upper bound on its cardinality, and shows how to construct a class of codes achieving this bound. Later, Gabidulin (1985) proved several properties and showed an efficient decoding algorithm. Given the significance of these contributions, this class of codes receives its name after the author. Roth (1991) also discovered this class of codes, independently of previous works.

This section starts defining the notions of rank weight and rank distance, shows that the rank distance induces a metric, and establishes some properties related to the rank of products involving matrices. Then, we define Gabidulin codes as the evaluation of linearized polynomials up to a certain degree, define their generator and parity-check matrices and give examples of the computation of a codeword and its syndrome.

Definition 2.12 (Rank weight)**.** Let $\mathbf{v} \in \mathbb{F}_{q^m}^n$ and let f_B denote the subfield map with respect to a basis B of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. The rank weight of $\mathbf{v}$ is defined as

$$\omega(\mathbf{v}) = \mathrm{rank}(f_B(\mathbf{v})).$$

Definition 2.13 (Rank distance)**.** Let the rank weight be defined as above. The rank distance between any two elements $\mathbf{u}, \mathbf{v} \in \mathbb{F}_{q^m}^n$ is given by

$$\delta(\mathbf{u}, \mathbf{v}) = \omega(\mathbf{u} - \mathbf{b}) = \mathrm{rank}(f_B(\mathbf{u}) - f_B(\mathbf{v})).$$

Definition 2.14 (Metric)**.** Let S be a set. The distance measure function δ applied on any two elements in S is a metric if, for any $a, b, c \in S$, the following properties are satisfied:

1. **Positive definiteness:** $\delta(a, b) \geq 0,$ with $\delta(a, b) = 0$ if and only if $a = b$.
2. **Symmetry:** $\delta(a, b) = \delta(b, a)$.
3. **Triangle inequality:** $\delta(a, b) + \delta(b, c) \geq \delta(a, c)$.

Given the definition of rank distance in Definition 2.13 and the requirements of a metric in Definition 2.14 we can prove rank distance is indeed a metric.

Lemma 2.2 (Rank distance induces a metric). Let $\mathbb{F}_{q^m}^n$ be an n-dimensional vector space over $\mathbb{F}_q$. Then, the rank distance function ω in Definition 2.13 induces a metric over $\mathbb{F}_{q^m}^n$.

Proof. Let $\mathbf{A}, \mathbf{B}, \mathbf{C}$ be the subfield mappings of any three elements of $\mathbb{F}_{q^m}^n$. By definition, the rank of a matrix is a non-negative integer. Moreover it is zero if and only if $\mathbf{A} - \mathbf{B} = \mathbf{0}$ or, equivalently, if and only if $\mathbf{A} = \mathbf{B}$, and we prove positive definiteness. Symmetry is trivial since $\mathrm{rank}(\mathbf{A}) = \mathrm{rank}(-\mathbf{A})$ and so it holds for $\mathbf{A} - \mathbf{B}$. For triangle inequality, we

rely on the well known property of linear algebra that

$$\operatorname{rank}(\mathbf{A}+\mathbf{B}) \leq \operatorname{rank}(\mathbf{A}) + \operatorname{rank}(\mathbf{B}),$$

then, we have

$$\begin{aligned}\operatorname{rank}(\mathbf{A}-\mathbf{C}) &= \operatorname{rank}(\mathbf{A}-\mathbf{B}+\mathbf{B}-\mathbf{C}) \\ &\leq \operatorname{rank}(\mathbf{A}-\mathbf{B}) + \operatorname{rank}(\mathbf{B}-\mathbf{C}),\end{aligned}$$

and the proof is complete. ■

Another property concerning the rank of matrices over finite fields and that will be important is the rank of the product between matrices (Lemma 2.3 or between a vector and a matrix with elements restricted to a particular subspace of the finite field (Lemma 2.4).

Lemma 2.3 (Rank of matrix product)*.* Let $\mathbf{A}$ and $\mathbf{B}$ be matrices of order $k \times m$ and $m \times n$, respectively. Them, the rank of $\mathbf{AB}$ is less than or equal to the minimum betweem the rank of $\mathbf{A}$ and the rank of $\mathbf{B}$. That is

$$\operatorname{rank}(\mathbf{AB}) \leq \min(\operatorname{rank}(\mathbf{A}), \operatorname{rank}(\mathbf{B})). \tag{4}$$

Lemma 2.4 (Rank of subspace matrix product)*.* Let $\mathbf{P}$ be an invertible matrix with entries in a δ-dimensional vector space $V \subset \mathbb{F}_{q^m}$. Then, for all $\mathbf{e} \in \mathbb{F}_{q^m}^n$,

$$\operatorname{rank}(\mathbf{eP}) \leq \delta \operatorname{rank}(\mathbf{e}). \tag{5}$$

Definition 2.15 (Gabidulin code)**.** Let $1 \leq k < n \leq m$ be integers, and $g_1, \ldots, g_n \in \mathbb{F}_{q^m}$ be linearly independent elements over $\mathbb{F}_q$. An (n, k) Gabidulin code over $\mathbb{F}_{q^m}$ defined at points $g_1, \ldots, g_n$ is the set of code words, each of which is defined as $\begin{pmatrix} p(g_1) & \ldots & p(g_n) \end{pmatrix}$, for a distinct linearized polynomial p over $\mathbb{F}_{q^m}$ of degree less than q^k. That is,

$$\operatorname{Gab}(n,k) = \left\{ \begin{pmatrix} p(g_1) & p(g_2) & \ldots & p(g_n) \end{pmatrix} \,:\, p(x) \in \mathbb{L}_{q^m},\ \deg(p(x)) < q^k \right\}. \tag{6}$$

In practice, the polynomials $p(x)$ correspond to the possible messages in the message space and will become clear when we present the generator matrix for a Gabidulin code and link its use to evaluation of a linearized polynomial at points $g_1, g_2, \ldots, g_n$.

Definition 2.16 (Generator matrix of a Gabidulin code)**.** Let $\mathcal{C}$ be an (n, k) Gabidulin code defined over points $g_0, \ldots, g_{n-1} \in \mathbb{F}_{q^m}$. Then, the generator matrix $\mathbf{G}$ of $\mathcal{C}$ has the following form

$$\mathbf{G} = \begin{pmatrix} g_1 & g_2 & \cdots & g_n \\ g_1^q & g_2^q & \cdots & g_n^q \\ \vdots & \vdots & \ddots & \vdots \\ g_1^{q^{k-1}} & g_2^{q^{k-1}} & \cdots & g_n^{q^{k-1}} \end{pmatrix}. \tag{7}$$

We call the vector $\mathbf{g} = \begin{pmatrix} g_1 & g_2 & \cdots & g_n \end{pmatrix}$, composed of the elements from the first row of $\mathbf{G}$, the generator vector of $\mathcal{C}$. It is not unique—nor the generator matrix—as any vector $\alpha\mathbf{g}$, with $\alpha \in \mathbb{F}_{q^m}$, is also a generator vector of $\mathcal{C}$. As an example, consider the $(4, 2)$ Gabidulin code $\mathcal{C}$ over $\mathbb{F}_2[x]/(x^4 + x + 1)$ defined at the points $x^2 + x$, $x^2 + 1$, $x^3 + x^2 + x$, and $x^3 + x^2 + x + 1$. Then, the generator matrix of $\mathcal{C}$ is

$$\mathbf{G} = \begin{pmatrix} x^2 + x & x^2 + 1 & x^3 + x^2 + x & x^3 + x^2 + x + 1 \\ x^2 + x + 1 & x & x^3 + x + 1 & x^3 + x \end{pmatrix}. \tag{8}$$

Lemma 2.5 (Parity-check matrix of a Gabidulin code)*.* Let $\mathcal{C}$ be an (n, k) Gabidulin code over $\mathbb{F}_{q^m}$ defined at points $g_1, g_2, \ldots, g_n$ and let $h_1, h_2, \ldots, h_n \in \mathbb{F}_{q^m}$ such that $\sum_{i=1}^{n} g_i^{q^j} h_i = 0$ for $j \in [k - n + 1, k - 1]$. The parity-check matrix $\mathbf{H}$ of $\mathcal{C}$ is defined as

$$\mathbf{H} = \begin{pmatrix} h_1 & h_2 & \cdots & h_n \\ h_1^q & h_2^q & \cdots & h_n^q \\ \vdots & \vdots & \ddots & \vdots \\ h_1^{q^{n-k-1}} & h_2^{q^{n-k-1}} & \cdots & h_n^{q^{n-k-1}} \end{pmatrix} \tag{9}$$

That is, the values h_i are a solution for the system of linear equations involving the points at which generated the code. In practice, if we let $\mathbf{G}$ be a generator matrix of $\mathcal{C}$, each of these equations corresponds to the calculation of an entry of the matrix $\mathbf{G}\mathbf{H}^\top$. Since the system if homogeneous, the matrix $\mathbf{H}$ is the parity-check matrix of $\mathcal{C}$. To ilustrate, take for instance, $j = k - n + 1$, obtaining the equation.

$$\sum_{i=1}^{n} g_i^{q^{k-n+1}} h_i = \sum_{i=1}^{n} \left(g_i^{q^{k-n+1}} h_i \right)^{q^{n-k-1}} = \sum_{i=1}^{n} g_i h_i^{q^{n-k-1}} = 0 \tag{10}$$

This corresponds exactly to the last element in the first row of $\mathbf{G}\mathbf{H}^\top$. Similarly, other values of j results in the equation for different entries of $\mathbf{G}\mathbf{H}^\top$. We now expand on the previous example, constructing the parity-check matrix of the code generated by $\mathbf{G}$ in Equation 8. First, we concoct the following system of linear equations in matrix form:

$$\begin{pmatrix} x^2+x^{2^{-1}} & x^2+1^{2^{-1}} & x^3+x^2+x^{2^{-1}} & x^3+x^2+x+1^{2^{-1}} \\ x^2+x^{2^0} & x^2+1^{2^0} & x^3+x^2+x^{2^0} & x^3+x^2+x+1^{2^0} \\ x^2+x^{2^1} & x^2+1^{2^1} & x^3+x^2+1^{2^1} & x^3+x^2+x+1^{2^1} \end{pmatrix} \begin{pmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{pmatrix} = \mathbf{0} \tag{11}$$

Solving the system, we obtain $h_0 = x^3 + 1$, $h_1 = x^2$, $h_2 = x^3 + x^2 + x$, and $h_3 = 1$. Now, we can construct the code's parity-check matrix:

$$\mathbf{H} = \begin{pmatrix} x^3+1 & x^2 & x^3+x^2+x & 1 \\ x^3+x^2+1 & x+1 & x^3+x+1 & 1 \end{pmatrix} \tag{12}$$

We finish this section on Gabidulin codes with an almost complete example of encoding and decoding. In reality what we do is to encode a message and then show that the syndrome of the resulting codeword is zero. To that, consider the code $\mathcal{C}$ and its generator and parity-check matrices $\mathbf{G}$ and $\mathbf{H}$ we have been working with.

Example 2.5.1 (Encoding a message with the generator matrix)**.** To encode a message $\mathbf{m} = \begin{pmatrix} x^2 & x \end{pmatrix}$ with $\mathcal{C}$ one multiplies it by the generator matrix $\mathbf{G}$:

$$\begin{aligned} \mathbf{mG} &= \begin{pmatrix} x^2 & x \end{pmatrix} \begin{pmatrix} x^2+x & x^2+1 & x^3+x^2+x & x^3+x^2+x+1 \\ x^2+x+1 & x & x^3+x+1 & x^3+x \end{pmatrix} \\ &= \begin{pmatrix} x^2+1 & x+1 & x^3 & x^3+x^2+x \end{pmatrix} \end{aligned} \tag{13}$$

That is, the linear transformation induced by $\mathbf{G}$ takes each message in $\mathbb{F}_{q^m}^k$ into a codeword in $\mathcal{C}$. Before showing an example involving the parity-check matrix, we establish the mapping between the evaluation of the linearized polynomial in Definition 2.15. For that, consider the message from the previous example. If we write it as the polynomial $p(y) = x \cdot y^2 + x^2 \cdot y$ over indeterminate x, and evaluate at $g_1 = x^2 + x$ from the previous example, we recover the first component of the codeword $\mathbf{mG}$:

$$\begin{aligned} x \cdot g_1^{2^1} + x^2 \cdot g_1^{2^0} &= x(x^2+x+1) + x^2(x^2+x) \\ &= x^2 + 1. \end{aligned}$$

Example 2.5.2 (Computing the syndrome with the parity-check matrix)**.** Let $\mathbf{c}$ be the codeword computed in Example 2.5.1. To calculate the syndrome of $\mathbf{c}$, we multiply

$$\mathbf{cH}^\top = \begin{pmatrix} x^2+1 \\ x+1 \\ x^3 \\ x^3+x^2+x \end{pmatrix}^\top \begin{pmatrix} x^3+1 & x^3+x^2+1 \\ x^2 & x+1 \\ x^3+x^2+x & x^3+x+1 \\ 1 & 1 \end{pmatrix} = \mathbf{0} \tag{14}$$

Since $\mathbf{c}$ belong the code $\mathcal{C}$, its calculated syndrome is $\mathbf{0}$. However, if we add and error $\mathbf{e}$ of rank weight 1, say $\begin{pmatrix} 0 & 0 & 0 & x \end{pmatrix}$, we end up with

$$\mathbf{cH}^\top = \begin{pmatrix} x^2+1 \\ x+1 \\ x^3 \\ x^3+x^2 \end{pmatrix}^\top \begin{pmatrix} x^3+1 & x^3+x^2+1 \\ x^2 & x+1 \\ x^3+x^2+x & x^3+x+1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} x & x \end{pmatrix} \tag{15}$$

3 RANK METRIC BASED CRYPTOSYSTEMS

3.1 INTRODUCTION

Cryptographic schemes and, more specifically, public-key encryption schemes based on rank metric codes started with the GPT cryptosystem in Gabidulin, Paramonov, and Tretjakov (1991), where the authors adapted the McEliece cryptosystem to Gabidulin codes. The peculiarities of this type of code allowed a myriad of attacks from day one, with the most devastating among them due to Overbeck. These days, even though techniques proposed to hinder the Overbeck's attack exist, new algebraic attacks pose a menace to these schemes. This chapter outlines the timeline of rank-based cryptosystems, introducing GPT, the attacks proposed, and the patches to the scheme to thwart attacks. Then, it finishes presenting in detail the PKE introduced in 2017 by Pierre Loidreau and its IND-CCA-secure variant by Shehhi et al.

The first public-key encryption scheme lying on the theory of error-correcting codes appeared in McEliece (1978). In this work, the author proposes an encryption/decryption scheme analogous to encoding/decoding but, to prevent unauthorized parties from accessing the messages, the ciphertext should look like a codeword from a random linear code, for which decoding is NP-complete (BERLEKAMP; MCELIECE; TILBORG, 1978). To this end, McEliece used a Goppa code and hid the generator matrix $\mathbf{G}$, multiplying it to the left by a matrix $\mathbf{S}$ and to the right by another matrix, $\mathbf{P}$. The resulting matrix $\mathbf{G}' = \mathbf{SGP}$ generates a linear code with the same minimum distance and rate as the one generated by $\mathbf{G}$. Then, the private key is the tuple $\mathbf{G}$, $\mathbf{S}$, $\mathbf{P}$, while the public key is $\mathbf{G}'$. On encryption, the user encodes a message $\mathbf{m}$ with $\mathbf{G}'$ and adds an error $\mathbf{e}$ of a certain distance t less than or equal to the error-correcting capacity of the code generated by $\mathbf{G}$. The decryption process multiplies the ciphertext by the inverse of $\mathbf{P}$, resulting in a codeword of the Goppa code generated by $\mathbf{G}$, namely $\mathbf{mSG} + \mathbf{eP}^{-1}$. This codeword is decoded into $\mathbf{mS}$ and multiplied by $\mathbf{S}^{-1}$, recovering the original message.

These schemes claim security against quantum computers by relying on similar problems that differ on the metric and the field over which the code is defined. Definition 3.1 states the rank metric version of it.

Definition 3.1 (Rank decoding problem)**.** Let $\mathbf{G} \in \mathbb{F}_{q^m}^{k \times n}$ be a matrix, $\mathbf{c} \in \mathbb{F}_{q^m}^{n}$ be a vector, and w an integer. Find $\mathbf{m} \in \mathbb{F}_{q^m}^{k}$ and $\mathbf{e} \in \mathbb{F}_{q^m}^{n}$ such that $\mathbf{c} = \mathbf{mG} + \mathbf{e}$ and the rank weight of $\mathbf{e}$ is less than or equal w?

Even if the rank decoding problem is not known to be NP-complete, there is a randomized reduction to an NP-complete problem (GABORIT; ZÉMOR, 2016). Next, we introduce the GPT cryptosystem.

3.2 THE GPT CRYPTOSYSTEM

The work in Gabidulin, Paramonov, and Tretjakov (1991) proposed a cryptosystem, named GPT in honor of its authors, based on McEliece's but this time using the so-called Gabidulin codes, introduced in the previous chapter. The argument is that decoding a codeword from a random code in rank metric is exponentially more difficult than decoding one from a random code in Hamming metric. See (GABORIT; RUATTA; SCHREK, 2016). Definition 3.2 below presents the GPT formally.

Definition 3.2 (GPT Cryptosystem)**.** The GPT cryptosystem is composed by a triple of probabilistic polynomial-time algorithms, (GEN, ENC, DEC), where:

GEN(1^N)
1 $n, k, r :=$ SELECTPARAMETERS(1^N)
2 $\mathbf{G} \leftarrow \{\mathbf{M} \in \mathbb{F}_{q^m}^{k \times n} \ : \ \langle \mathbf{M} \rangle \in \mathrm{Gab}(n, k)\}$
3 $\mathbf{S} \leftarrow \{\mathbf{M} \in GL_k(\mathbb{F}_{q^m})\}$
4 $\mathbf{a} \leftarrow \{\mathbf{v} \in \mathbb{F}_{q^m}^k \ : \ \mathbf{v} \neq \mathbf{0}\}$
5 $\mathbf{e}_0 \leftarrow \{\mathbf{v} \in \mathbb{F}_{q^m}^n \ : \ 0 < \omega(\mathbf{v}) \leq r\}$
6 $\mathbf{G}' := \mathbf{SG} + \mathbf{a}^\top \mathbf{e}_0$
7 **return** $pk = \mathbf{G}', \ sk = (\mathbf{G}, \mathbf{S}^{-1})$

Algorithm 1: GPT key generation

ENC$_{pk}(\mathbf{m})$
1 $\mathbf{e} \leftarrow \{\mathbf{v} \in \mathbb{F}_{q^m}^n \ : \ \omega(\mathbf{v}) = \lfloor \frac{n-k}{2} \rfloor - r\}$
2 $\mathbf{c} := \mathbf{mG}' + \mathbf{e}$
3 **return** $\mathbf{c}$

Algorithm 2: GPT encryption

DEC$_{sk}(\mathbf{c})$
1 $\mathbf{c}_0 :=$ DECODE$(\mathbf{c}, \mathbf{G})$
2 $\mathbf{m} := \mathbf{c}_0 \mathbf{S}^{-1}$
3 **return** $\mathbf{m}$

Algorithm 3: GPT decryption

Before explaining the algorithms, we point to the subroutine SELECTPARAMETERS in Step 1 of Algorithm 1. We use this subroutine to indicate the selection of arbitrary parameters, usually the ones suggested by the authors. So, in Algorithm 1, for example, N may be the scheme security in bits and the parameters k, m, and n the suggested values that attain the N bits security. The subroutine DECODE in Algorithm 3 stands for any polynomial-time decoder for a Gabidulin code taking as input a codeword in the code and the generator matrix of this code, even though particular instances take different parameters.

The key generation in the GPT cryptosystem starts with parameters k, m, n, and r. It uses the first three to construct a Gabidulin code; they must obey the relation $k < n \leq m$. The last parameter, r, must be less than the error-correcting capacity of the code used in the scheme as we will see later. Here we emphasize that the original GPT cryptosystem uses $n = m$, while Definition 3.2 uses a more general Gabidulin code, that is, $n \leq m$. We make this choice because later schemes use values of n that may not be equal to m.

Step 2 generates a random matrix $\mathbf{G}$ that generates an (n, k) Gabidulin code over $\mathbb{F}_{q^m}$. This matrix is part of the secret key, and the subsequent steps will transform it into a generator matrix of a random-looking code. To that, Steps 3, 4, and 5 generate a random invertible matrix $\mathbf{S}$ over $\mathbb{F}_{q^m}$, of order k, and random vectors $\mathbf{a}$ in $\mathbb{F}_{q^m}^k$ and $\mathbf{e}_0$ in $\mathbb{F}_{q^m}^n$, such that the former is different from $\mathbf{0}$ and the latter has rank weight less than or equal to r. Finally, Step 6 computes the public code generator matrix $\mathbf{G}'$, and Step 7 returns $\mathbf{G}'$ and $(\mathbf{G}, \mathbf{S})$ as the public and private keys, respectively.

Contrary to the original McEliece scheme, the key generation in GPT does not multiply the code generator matrix $\mathbf{G}$ to the right by an invertible matrix. Instead, it adds $\mathbf{SG}$ to a random matrix that decomposes into two vectors, as specified in Algorithm 1. This way, encoding a message $\mathbf{m}$ with $\mathbf{G}'$ results in $\mathbf{mSG} + \mathbf{ma}^\top \mathbf{e}_0$. The left-hand side of this addition is a codeword in the secret code, namely the encoding of a message $\mathbf{mS}$ using $\mathbf{G}$ and the right-hand side is a vector of rank weight at most r. To see this, consider the matrices $\mathbf{A}$ and $\mathbf{B}$, subfield mappings of $\mathbf{ma}^\top$ and $\mathbf{e}_0$, respectively, for some basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. Given Lemma 2.3, we have $\mathrm{rank}(\mathbf{AB}) \leq \min(\mathrm{rank}(\mathbf{A}), \mathrm{rank}(\mathbf{B})) \leq r$. In practice, the matrix $\mathbf{G}'$ generates a code with a smaller error-correcting capacity than the code generated by $\mathbf{G}$ and, from the coding theory point of view, the addition of $\mathbf{a}^\top \mathbf{e}_0$ corresponds to adding an error that depends on the value of $\mathbf{m}$.

Encryption (Algorithm 2) and decryption (Algorithm 3) of the GPT cryptosystem are straightforward. The first encodes the input plaintext with $\mathbf{G}'$ and adds a random error $\mathbf{e}$, which rank summed to r, the maximum possible rank of $\mathbf{e}_0$, equals the number of errors one can correct using $\mathbf{G}$. The second decodes the ciphertext it receives as a parameter using $\mathbf{G}$ and recovers the original plaintext by multiplying the result by $\mathbf{S}^{-1}$ to the right.

The GPT system suffered its first attack in Gibson (1995) and Gibson (1996), where the cryptanalytic algorithm presented runs in exponential time and only worked for the small parameters proposed at that time. Latter works introduced variants of GPT that attempted to better hide the structure of the secret code. In Gabidulin and Ourivski (2001) and Gabidulin, Ourivski, et al. (2003) the authors returned to the idea of using a column scrambler matrix $\mathbf{P}$ as in the original McEliece proposal to avoid Gibson's attack. However, Overbeck (2005) and Overbeck (2008) completely broke the system using the strong structure of Gabidulin codes and the fact $\mathbf{P}$ is defined over $\mathbb{F}_q$. Even subsequent schemes claiming security against the Overbeck attack by taking the elements from

$\mathbb{F}_{q^m}$ instead of $\mathbb{F}_{q^m}$, such as Gabidulin (2008), Gabidulin, Rashwan, and Honary (2009), and Rashwan, Gabidulin, and Honary (2011), could not stand. Otmani, Kalachi, and Ndjeya (2018) have shown latter that is possible to reformulate all of them as instances of the system attacked by Overbeck. The solution to this problem came only in Loidreau (2017) which mixed the ideas usually applied in McEliece-like cryptosystems to others ideas arising from the design of low rank parity-check codes (LRPC).

3.3 THE LOIDREAU CRYPTOSYSTEM

The main weakness of variants of the GPT cryptosystem exploited in the works Overbeck (2005), Overbeck (2008), and Otmani, Kalachi, and Ndjeya (2018) is the possibility to express the public code generator $\mathbf{G}'$ of each of these encryption schemes as a matrix $\mathbf{S}\begin{pmatrix}\mathbf{X} & \mathbf{G}\end{pmatrix}\mathbf{P}$, where $\mathbf{P}$ is a square matrix over the ground field $\mathbb{F}_q$ of order $(n+t)$, and $\mathbf{S}$, $\mathbf{G}$, and $\mathbf{X}$ are matrices over $\mathbb{F}_{q^m}$ of dimensions $s \times k$, $k \times n$, and $k \times t$, respectively. Then, an attacker can apply the operator Λ_i to the public code generator

$$\Lambda_i(\mathbf{G}') = \begin{pmatrix} \mathbf{G}' \\ (\mathbf{G}')^q \\ \vdots \\ (\mathbf{G}')^{q^i} \end{pmatrix}, \tag{16}$$

where $(\mathbf{G}')^{q^i}$ is the matrix $\mathbf{G}'$ with each of its elements raised to the power q^i.

While for a matrix $\mathbf{M}$ generating a random $(n+t, s)$ code over $\mathbb{F}_{q^m}$ the rank of $\Lambda_i(\mathbf{M})$ would be, with great probability, $\min(n+t, s(i+1))$, for the public code generator $\mathbf{G}'$ the rank of the result after applying Λ_i would be $\min(n+t, s+t+i)$. This clearly distinguishes the code generated by $\mathbf{G}'$ from a random code. Moreover, under certain circumstances, the attacker can find an alternative column scrambler $\mathbf{P}'$ and thus recover a valid secret key in $O(n^3)$ using elementary linear algebra.

Observing this fact, Loidreau proposed the cryptosystem in Definition 3.3, in which the column scrambler draws its elements neither from $\mathbb{F}_q$ or $\mathbb{F}_{q^m}$. Rather, they belong to a vector subspace of small dimension contained in the extension field $\mathbb{F}_{q^m}$ seen as an m-dimensional vector space over $\mathbb{F}_q$.

Definition 3.3 (Loidreau Cryptosystem)**.** The Loidreau cryptosystem is composed by a triple of probabilistic polynomial-time algorithms, (GEN, ENC, DEC), where:

GEN(1^N)

1 $k, m, n, \delta :=$ SELECTPARAMETERS(1^N)
2 $\mathbf{G} \leftarrow \{\mathbf{M} \in \mathbb{F}_{q^m}^{k \times n} \ : \ \langle \mathbf{M} \rangle \in \mathrm{Gab}(n, k)\}$
3 $\mathbf{S} \leftarrow \{\mathbf{M} \in GL_k(\mathbb{F}_{q^m})\}$
4 $\mathbf{P} \leftarrow \{\mathbf{M} \in GL_n(V) \ : \ V \subset \mathbb{F}_{q^m}$ **and** $\dim(V) = \delta\}$
5 $\mathbf{G}' := \mathbf{SGP}^{-1}$
6 **return** $pk = \mathbf{G}', \ sk = (\mathbf{G}, \mathbf{S}^{-1}, \mathbf{P})$

Algorithm 4: Loidreau key generation

ENC$_{pk}(\mathbf{m})$

1 $\mathbf{e} \leftarrow \{\mathbf{v} \in \mathbb{F}_{q^m}^n \ : \ \omega(\mathbf{v}) = \lfloor \frac{n-k}{2\delta} \rfloor\}$
2 $\mathbf{c} := \mathbf{mG}' + \mathbf{e}$
3 **return** $\mathbf{c}$

Algorithm 5: Loidreau encryption

DEC$_{sk}(\mathbf{c})$

1 $\mathbf{c}_0 := \mathbf{cP}$
2 $\mathbf{c}_1 :=$ DECODE$(\mathbf{c}_0, \mathbf{G})$
3 $\mathbf{m} := \mathbf{c}_1\mathbf{S}^{-1}$
4 **return** $\mathbf{m}$

Algorithm 6: Loidreau decryption

From Definition 3.2 the main difference relates to the key generation algorithm GEN. In Step 1 of Algorithm 4, SELECTPARAMETERS outputs an extra parameter δ, which is the dimension of the subspace containing the elements of $\mathbf{P}$. It is important to choose δ so that the code rate is greater than or equal to $1 - \frac{1}{\delta}$ and that δ itself is greater than 3 as Coggia and Couvreur (2020), Ghatak (2022), and Loidreau and Pham (2022) demonstrated that a distinguisher, and consequently a secret key, can be recovered for parameter sets not obeying these conditions. With δ selected, the only remaining difference is in Step 4 where rather than constructing a matrix using two vectors and adding $\mathbf{SG}$, as in Algorithm 1, it generates a random δ-dimensional vector subspace V and constructs an invertible matrix $\mathbf{P}$ from random elements in V. $\mathbf{P}$ becomes part of the private key and its inverse multiplies $\mathbf{SG}$ to the right to produce the public code generator $\mathbf{G}'$ as in Step 5.

To encrypt a message $\mathbf{m}$, Algorithm 5 does exactly what the encryption algorithm in GPT does, except that the rank weight of the generated error should be $\lfloor \frac{n-k}{2\delta} \rfloor$. Then, to decrypt a ciphertext $\mathbf{c}$, instead of decoding it directly using $\mathbf{G}$, it first multiplies $\mathbf{c}$ by the private key component $\mathbf{P}$ and only in Step 2 it decodes the result $\mathbf{c}_0 = \mathbf{mSG} + \mathbf{eP^{-1}}$ of Step 1. This step explains the reason behind the choice of the rank weight in Step 1 of ENC. Given Lemma 2.4, we have $\omega(\mathbf{eP}) \leq \delta \lfloor \frac{n-k}{2\delta} \rfloor \leq \lfloor \frac{n-k}{2} \rfloor$, and so it can be decoded

using $\mathbf{G}$. After that, the algorithm follows in the same manner as in GPT and outputs the decrypted message.

The argument the author gives for the security of its cryptosystem is that elements in $\mathbf{P}$ belong to the vector subspace V of $\mathbb{F}_{q^m}$, but the same is not guaranteed for the elements in $\mathbf{P}^{-1}$. After all, even if we draw the elements from a specific subspace (V in this case) the operations are still performed in $\mathbb{F}_{q^m}$. Therefore, applying the operator Λ_i to $\mathbf{G}'$, as constructed in Step 5 of Algorithm 4, changes the rank of the resulting matrix in a manner that the distinguishing property used in Overbeck attack no longer holds, i.e., it is not possible to distinguish the public code generator matrix from a random rank metric code generator. It seems this proposal solves the problem of structural attacks for schemes based on Gabidulin codes. However, to be practical, an encryption scheme needs to offer security guarantees such as the ones we presented in Section 2.2.1 and, as the author states, the Loidreau cryptosystem does not achieve indistinguishability; neither against chosen-plaintext or chosen-ciphertext attacks. In the next section we present an encryption scheme that solves this issue.

3.4 THE SHEHHI ET AL. PUBLIC-KEY ENCRYPTION SCHEME

The encryption scheme proposed in Loidreau (2017) does not achieve indistinguishability under chosen-plaintext attack, which is the minimum required by the NIST competition. Therefore, it does not provide security against chosen-ciphertext attack as well. A security notion required in many use cases. To see why this is the case, let Π be an instance of the Loidreau PKE as in Definition 3.3 and A an attacker that in Step 2 of the chosen-plaintext indistinguishability experiment $\mathrm{EXP}^{\mathrm{exp}}_{\mathrm{A},\Pi}$, select messages $\mathbf{m}_0 = \mathbf{0}$ and $\mathbf{m}_1 \neq \mathbf{0}$. If the challenge ciphertext $\mathbf{c}$ is the encryption of $\mathbf{m}_0$, for sure its rank weight is $\lfloor \frac{n-k}{2\delta} \rfloor$ while if it is the encryption of $\mathbf{m}_1$ there is a great chance for its rank to be different from $\lfloor \frac{n-k}{2\delta} \rfloor$. This is enough for an attacker to guess the value of b in Step 5 of the experiment with probability greater than $1/2$.

Given that, Shehhi et al. (2019) proposed an IND-CCA variant of Loidreau's cryptosystem borrowing ideas presented in works such as Fujisaki and Okamoto (1999), Fujisaki and Okamoto (2013), Hofheinz, Hövelmanns, and Kiltz (2017), and Saito, Xagawa, and Yamakawa (2018), usually applied to a public key encryption scheme to turn it into an IND-CCA key encapsulation mechanism or hybrid encryption scheme. The product of the transformation presented by the authors is nonetheless a PKE scheme.

The new construction, presented in Definition 3.4, demands the use of two hash functions. One is a traditional hash function, referred to as H_0. The other, H_1, is a hash function where one can choose the output size via an extra argument, called an extendable-output function (XOF). It is important to note these functions need to offer at least the same level of security expected for the PKE as a whole. Otherwise its security is reduced to the hash algorithms.

The transformations from previous works normally require the decryption algorithm to recompute the ciphertext, i.e., repeating the encryption process with the decrypted message as input and adding the error recovered in the decoding phase instead of a fresh generated one. This is not necessary in this IND-CCA variant and it could even spare a matrix multiplication if was not for the need to recover the error from $\mathbf{eP}^{-1}$ in decryption. Nevertheless, the result from Shehhi et al. (2019) shows that the decryption in their implementation of the scheme is about 5% slower in comparison to the implementation of Loidreau's in Abdouli et al. (2018) corresponding to the extra matrix multiplication. This overhead is comparable to the application of a transformation like Fujisaki-Okamoto alone, excluding particularities of the PKE it is applied to. On the encryption side the lost of performance is more significant, 23%, since computing the two digests occupies a large portion of time compared to the matrix multiplication to encode the message and some additions that can be done via XOR operations.

The transformation proposed by the authors allows one to encrypt an amount of data larger than the Fujisaki-Okamoto's at a cost of 23% in decryption only. It can be employed, for example, as KEM to exchange multiple keys in one ciphertext. The security proof in Shehhi et al. (2019) relies on properties specific to the Loidreau's scheme. Nevertheless, the authors claim that the transformation might be adapted to other schemes as well. We now define the Shehhi et al. cryptosystem.

Definition 3.4 (Shehhi et al. Cryptosystem)**.** The Shehhi et al. cryptosystem is composed by a triple of probabilistic polynomial-time algorithms, (GEN, ENC, DEC), where:

GEN(1^N)

1 $k, m, n, \delta := \text{SELECTPARAMETERS}(1^N)$
2 $\mathbf{G} \leftarrow \{\mathbf{M} \in \mathbb{F}_{q^m}^{k\times n} \ : \ \langle \mathbf{M} \rangle \in \mathrm{Gab}(n,k)\}$
3 $\mathbf{S} \leftarrow \{\mathbf{M} \in GL_k(q^m)\}$
4 $\mathbf{P} \leftarrow \{\mathbf{M} \in GL_n(V) \ : \ V \subset \mathbb{F}_{q^m}$ **and** $\dim(V) = \delta\}$
5 $\mathbf{G}' := \mathbf{SGP}^{-1}$
6 **return** $pk = \mathbf{G}', \ sk = (\mathbf{G}, \mathbf{S}^{-1}, \mathbf{P})$

Algorithm 7: Shehhi et al. key generation

ENC$_{pk}(\mathbf{m})$

1 $\mathbf{e} \leftarrow \left\{\mathbf{v} \in \mathbb{F}_{q^m}^n : \omega(\mathbf{v}) = \left\lfloor \frac{n-k}{2\delta} \right\rfloor\right\}$
2 $\mathbf{v} := \mathsf{H}_0(\mathbf{e}, \mathbf{m})$
3 $\mathbf{m}_0 := (\mathbf{m} \quad \mathbf{v})$
4 $\mathbf{m}_1 := \mathbf{m}_0 + \mathsf{H}_1(\mathbf{e})$
5 **return** $\mathbf{m}_1\mathbf{G}' + \mathbf{e}$

Algorithm 8: Shehhi et al. encryption

$\textsc{Dec}_{sk}(\mathbf{c})$
1 $\mathbf{c}_0 := \textsc{Decode}(\mathbf{cP}, \mathbf{G})$
2 **if** $\mathbf{c}_0 = \perp$
3 **return** $\perp$
4 $\mathbf{m}'_1 := \mathbf{c}_0\mathbf{S}^{-1}$
5 $\mathbf{e}' := \mathbf{c} - \mathbf{c}_0\mathbf{P}^{-1}$
6 $\mathbf{m}'_0 := \mathbf{m}'_1 - \mathsf{H}_1(\mathbf{e}')$
7 $(\mathbf{m}' \quad \mathbf{v}') := \mathbf{m}'_0$
8 **if** $\mathsf{H}_0(\mathbf{e}', \mathbf{m}') \neq \mathbf{v}'$ **and** $\omega(\mathbf{e}') \neq \lfloor \frac{n-k}{2\delta} \rfloor$
9 **return** $\perp$
10 **return** $\mathbf{m}'$

Algorithm 9: Shehhi et al. decryption

The key generation algorithm GEN in Definition 3.4 works exactly like the homonym algorithm in Definition 3.3. The difference between the two encryption schemes starts in ENC, where the plaintext $\mathbf{m}$ goes through a few transformations before encoding and adding the error vector $\mathbf{e}$ generated in Step 1. So, after generating $\mathbf{e}$ with prescribed rank weight, both the plaintext and the error are used as input to a hash function H_0. The resulting digest $\mathbf{v}$ is called the verification digest or verification hash, and it serves as a random parameter for correctness validation during decryption. Next, it augments the original ciphertext with $\mathbf{v}$ producing the extended plaintext $\mathbf{m}_0$ in Step 3. Following this, the extended plaintext is added to the digest of $\mathbf{e}$ using the extendable-output function H_1. Here it is imperative to set the output size of H_1 to match the size of $\mathbf{m}_0$. Lastly, Step 5 encrypts the result from Step 4 in the same manner as in Loidreau's PKE and returns the ciphertext.

The decryption of a ciphertext $\mathbf{c}$ described in Algorithm 7 works as follows. The first step is identical to that of Algorithm 6. That is, multiply the ciphertext to the right by $\mathbf{P}$ and then decode the result using an algorithm that decodes codewords in the code generated by $\mathbf{G}$. Step 2 checks if decoding was not successful, that is, if $\mathbf{c}_0$ is the decoding failure symbol $\perp$. If so, DEC immediately forwards the symbol in Step 3 to indicate a decryption failure and finishes. If not, the process continues, and Step 4 computes $\mathbf{m}'_1$ using the usual technique of multiplying by the inverse of the row scrambler matrix $\mathbf{S}$. Steps 1–4 are roughly equivalent to decryption in the Loidreau scheme. The following steps are specific to this algorithm.

Step 5 of Algorithm 9 retrieves an error vector $\mathbf{e}'$ that ENC supposedly used to generate the ciphertext. To visualize what is happening in this step, remember that if $\mathbf{c} = \mathbf{m}_1\mathbf{SGP}^{-1} + \mathbf{e}$, then $\mathbf{c}_0 = \mathbf{m}_1\mathbf{SG}$, and their difference equals the error $\mathbf{e}$. The digest of $\mathbf{e}'$, obtained by applying the extendable-output function H_1, is subtracted from $\mathbf{m}'_1$ to produce an extended plaintext $\mathbf{m}'_0$ in Step 6, and Step 7 splits it into two vectors, $\mathbf{m}'$ and $\mathbf{v}$, effectively undoing Step 3 of ENC. At this point, ENC has decrypted $\mathbf{c}$ into

the plaintext $\mathbf{m}'$. A check is necessary to confirm that $\mathbf{c}$ is a valid ciphertext, so Step 8 verifies if either the output of H_0 on inputs $\mathbf{e}'$ and $\mathbf{m}'$ does not match the verification hash $\mathbf{v}'$ or the rank weight of $\mathbf{e}'$ is different from the error-correcting capacity of the public code. If any of these inequalities hold, the algorithm stops returning the failure symbol. Otherwise, the decryption process is successful and returns the plaintext $\mathbf{m}'$.

3.5 ANALYSIS OF THE SECURITY OF RANK BASED CRYPTOSYSTEMS

Now we analyze rank-based encryption schemes concerning their security, key sizes, and code rates. To this end, we compute the theoretical complexity of recent attacks to our cryptosystem and depict how these attacks have affected the choice of parameters over the last years. We also compare the best attack using different techniques against rank-based systems.

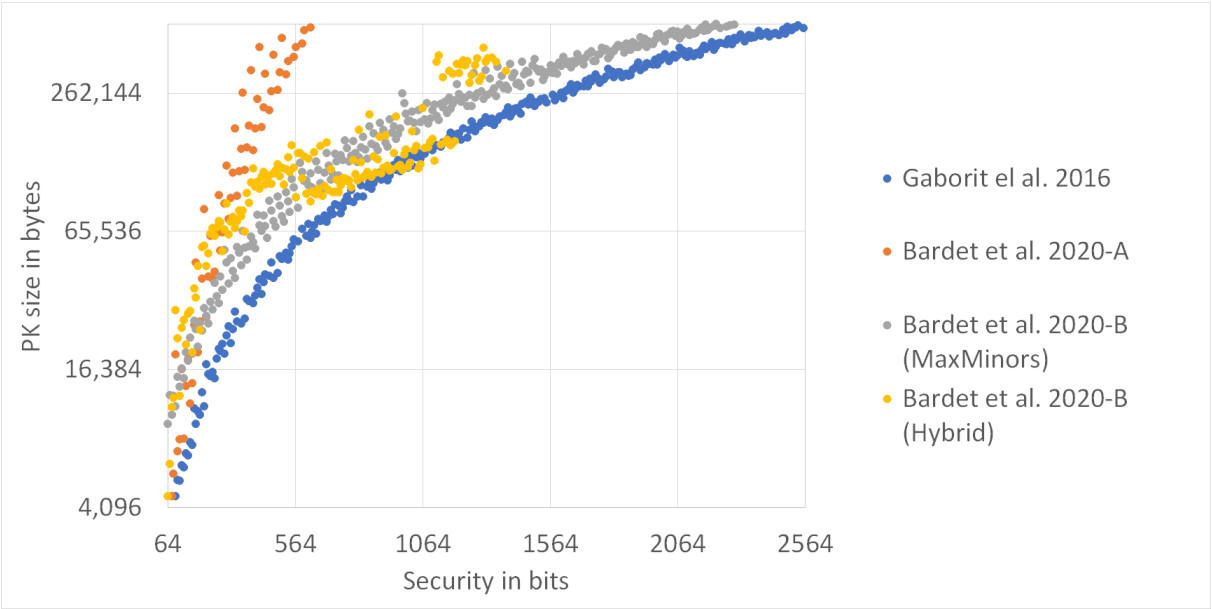
There are two possible forms of breaking a scheme based on rank metric codes. The first is to recover a decoder for the underlying code, the class to which the Overbeck attack mentioned earlier belongs. We disregard this attack as recent proposals seem to hinder this menace. However, for the Loidreau cryptosystem specifically, the Coggia and Covreur attack and its extensions restrict the choice of the subspace dimension δ; the designer has to make a trade-off between the private key size and the amount of redundant information in each ciphertext. Another form is to recover the message from the ciphertext via a combinatorial or algebraic approach.

Combinatorial attacks have been studied for longer, with Aragon et al. (2018) and Gaborit, Ruatta, and Schrek (2016) the most remarkable works and although these attacks can make use of Grover's algorithm to speed up, they are not as dangerous as the recent algebraic attacks. (GABORIT; RUATTA; SCHREK, 2016; BARDET; BRIAUD, et al., 2020; BARDET; BROS, et al., 2020) accelerated an area in which relevant discoveries did not show since Ourivski and Johansson (2002). Even if these attacks solve the rank decoding problem in exponential time, the advance was enough to left schemes such as ROLLO and RQC out of NIST's third round.

From Figure 3, it is easy to note that the attack in Bardet, Briaud, et al. (2020) needs much larger public-key sizes than the other as the security increases. However, Figure 4 focuses on the security levels relevant to use in practice, and it is possible to observe a different scenario. For them, the two attacks in Bardet, Bros, et al. (2020) force the scheme to have larger sizes in their public keys than the one in Bardet, Briaud, et al. (2020).

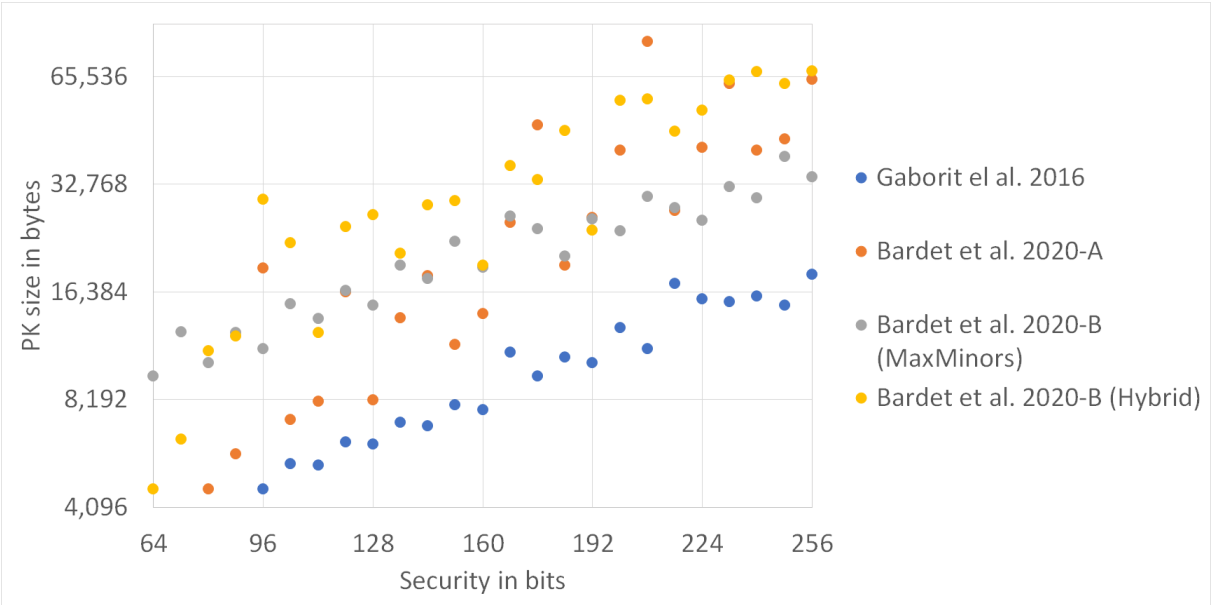
A feature noticed through our experiments is that a significant variation occurs in the sizes of keys offering the same level of security, even when considering a single attack. As we can see in the graph of Figure 4, on the line of 96 bits of security, there is a set of parameters whose public key must have about 30 KB to offer approximately that security against hybrid attack in Bardet, Bros, et al. (2020). Nevertheless, it is possible

Figure 3 – Minimum key sizes to resist algebraic attacks



Source: The author.

Figure 4 – Key sizes of Figure 3 for practical security levels



Source: The author.

to choose parameters that n way that the resulting key is shorter and more secure, such as the one in the 160-bit line, with approximately 25 KB. This observation highlights the importance of carefully choosing the parameters of our system.

We finish this chapter defining the parameters used in our implementation.Table 4 shows the parameters m, n, k, and δ, and the public and private keys sizes, for the more widely used security levels.

Table 4 – Selected parameters.

Security	m	n	k	δ	r	**PK size**	**SK size**
80	72	68	38	3	5	10260	15369
128	80	80	44	3	6	15840	22590
192	104	100	50	3	8	32500	37589
256	168	116	60	3	9	70560	83145

Source: The author.

Note: The key sizes are expressed in bytes.

Part II

Implementation

4 IMPLEMENTATION

4.1 SYSTEM OVERVIEW

Our implementation, available at github.com/flpborba/shehhi, uses Python, an interpreted high-level language. Python's dynamic type system together with its simple yet elegant syntax allows the rapid development of applications in a clear and organized manner. The language constructs enable a variety of programming styles, also called programming paradigms, such as procedural and object-oriented programming. These features make the language a reasonable choice for the purposes of this work.

Besides the advantages mentioned, the main reason for choosing Python as the implementation language is the use of SageMath (or Sage for short), which is an open-source mathematical software system providing the vast majority of the requirements for the construction of our cryptosystem, including Gabidulin codes, finite fields, and matrices. Another library we use is PyCryptodome, mainly for its implementations of hash algorithms, namely the SHA-3 family of algorithms and the extendable-output functions SHAKE128 and SHAKE256. PyCryptodome also offers functionality used to encode and decode keys in PEM and DER formats. The use of this libraries is expressed in the context model in Figure 5.

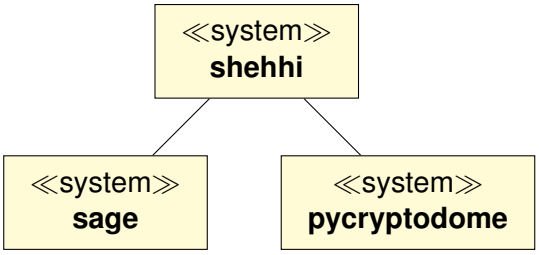


Figure 5 – Context model

The implementation is organized into the five packages depicted in Figure 6 according to their concerns. The `matrix` package implements functions to create random matrices and vectors with the necessary properties for use in our cryptosystem (e.g., a vector with a certain rank to use as the error in encryption). The `io` package contains functions to encode into bytes finite field elements, as well as matrices and vectors over finite fields. It also offers functions to decode bytes back into the mentioned objects.

The `hash` package gives the abstract classes `HashFunction` and `ExtendableOutputFunction`, which the hash functions and extendable-output functions used with the cryptosystem shall inherit. None of these classes imposes any restriction on the representation or construction of objects, except for the `__call__` method. For a class inheriting from `HashFunction`, it takes the message bytes and returns the message digest. For a class inheriting `ExtendableOutputFunction`, in addition to the message bytes, the output size is required. Another method of compulsory implementation for

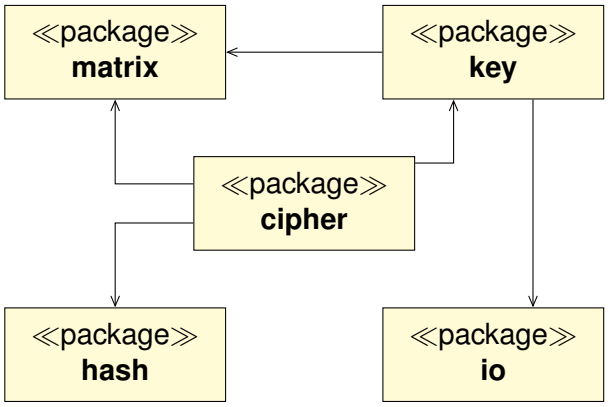


Figure 6 – Package organization

a `HashFunction` is `digest_size` that returns the number of bytes the message digest contains. The package also provides concrete implementations of both cryptographic primitives via classes `SHA3` and `SHAKE`, respectively (SHA-3 and SHAKE are well-known algorithms standardized by NIST).

Package `key` provides functionalities to handle private and public keys. Among these are the generation of private keys, the derivation of public keys from privates ones, and the importation and exportation of both types. The `generate` function generates a new `SecretKey` instance given the required security level (128, 192, or 256-bits). The actual values of m, n, k, and δ used for each security level are the ones in Shehhi et al. (2019). It is also possible to create a `SecretKey` directly, although not recommended. After creating an instance, one can use it to derive the corresponding `PublicKey` using the `public_key` method. We decided to separate the generation of private and public keys, so we generate the latter only when necessary (even if, in most cases, one would want to send it to another party or a repository).

Importation and exportation of keys support DER and PEM formats. The DER format encodes keys using the ASN.1 structure presented in the next section and returns the bytes, while the PEM format first encodes using DER and only then encodes the resulting bytes as the corresponding type of key. The package asymmetrically provides the two operations (i.e., class methods provide exportation since an instance is necessary while functions provide importation as we do not have the key yet).

The last package, `cipher`, is the one that glues it all together. It comprises two functors, `Enc` and `Dec`, that share the class `Cipher` as a common base. As expected, `Enc` and `Dec` implement Algorithm 8 and Algorithm 9, respectively, which implementations we will explain in detail in the next section. One creates an instance of one of these functors passing the proper key, a hash function, and an extendable-output function. As already mentioned, both function objects must inherit from the proper classes in the hash package. Notice that the cryptosystem has its security limited by the key and the hash algorithms used. Therefore, if one generates keys with 192 bits of security, these must be used with hash algorithms providing at least 192 bits of security.

4.2 PUBLIC AND PRIVATE KEYS

This section explains how keys are stored. We begin showing the ASN.1 syntax for our public and private keys, a standard description language used for data structures. Then, we relate the key generation in Algorithm 7 to the ASN.1 structure of the keys, explaining how these are generated, stored, and used in practice to reduce their size.

4.2.1 ASN.1 Syntax

The ASN.1 syntax is a joint standard of ISO/IEC and the ITU-T (International Telecommunication Union Telecommunication Standardization Sector). This standard aims to define a language to describe data structures independent of platform and is both human-readable and machine-readable. Frame 1 shows the ASN.1 syntax for public and private keys of our implementation. We will not discuss the syntax, nonetheless, to give an intuition of how it works, consider `PrivateKey` in the first line of Frame 1. This is a type that specifies how to encode private keys and is defined as a `SEQUENCE`, an ASN.1 type used to express a collection of grouped items. The items, enclosed in curly braces, are the components of `PrivateKey` and have an identifier and a type. For example, `generator` identifies an item that must be encoded as an `OCTET STRING` (i.e., binary data that is multiple of 8 bits). While the first three components of `PrivateKey` are of type `OCTET STRING`, a native ASN.1 type, the last item, `parameters`, is a type we defined ourselves.

```
PrivateKey ::= SEQUENCE {
    generator             OCTET STRING
    rowScrambler          OCTET STRING
    colScramblerBasis     OCTET STRING
    colScramblerSubvector OCTET STRING
    parameters            Parameters
}

PublicKey ::= SEQUENCE {
    publicGenerator BIT STRING
    parameters      Parameters
}

Parameters ::= SEQUENCE {
    extDegree   INTEGER
    codeLength  INTEGER
    codeDim     INTEGER
    subspaceDim INTEGER
}
```

Frame 1 – ASN.1 syntax of private and public keys

4.2.2 Encoding PKE Parameters

We start by type `Parameters`, the simplest of the three `SEQUENCE`s in Frame 1, containing four `INTEGER`s which are the parameters chosen in step:shehhi-Step 1 of Algorithm 7. The components `extDegree`, `codeLength`, `codeDim`, and `subspaceDim`, map to m, n, k, and δ, respectively. They are necessary to store because we use them to reconstruct the key parts, i.e., vectors and matrices, from data on disk. Next, we present `PrivateKey`. and explain how it differs from the key presented in Algorithm 7 and how it is generated in practice to reduce size in storage.

4.2.3 Encoding Keys

Component `generator` is simple. The private key contains the matrix $\mathbf{G}$ generated in Algorithm 7, or as we will see later, a Gabidulin code with such a generator. However, instead of storing the matrix in its entirety, we keep only its first row, which corresponds to the code generator vector. Later, when we want to use the key again, it is easy to reconstruct $\mathbf{G}$ by observing Definition 2.16. This method reduces the size needed to store $\mathbf{G}$ by a factor of k.

The matrix $\mathbf{P}$ from GEN maps to components `colScramblerBasis` and `colScramblerSubvector`. If we consider that each element of $\mathbb{F}_{2^m}$ needs m bits of storage, $\mathbf{P}$ would typically require mn^2 bits. But the entries on it are taken from a δ-dimensional subspace V of $\mathbb{F}_{2^m}$, so we take advantage of it to reduce the necessary disk space by storing a matrix and a vector. Let $\mathbf{A}$ and $\mathbf{B}$ represent `colScramblerSubvector` and `colScramblerBasis`, respectively. The latter is the basis of V and has dimension $m \times \delta$, and the former is an $n \times n$ matrix over $\mathbb{F}_{2^\delta}$ where, for each element a_{ij}, the linear combination of rows of $\mathbf{B}$ given by the coefficients of a_{ij} results in the element p_{ij} of P. That is, for each $a_{ij} \in \mathbf{V}$,

$$a_{i,j}\mathbf{B} = p_{i,j} \in \mathbf{P}.$$

In fact, this is how the implementation generates $\mathbf{P}$, with a random basis of V and a random matrix with elements in $\mathbb{F}_{2^\delta}$. Storing it in this way reduces the size required to store the column scrambler matrix to $\delta(m + n^2)$ and the reduction is not negligible since, when comparing the dimensions of the three matrices $\mathbf{G}$, $\mathbf{S}$, and $\mathbf{P}$, the latter is responsible for the largest storage space. To make things concrete, consider parameters $m = 128$, $n = 120$, $k = 80$, and $\delta = 5$ proposed in Loidreau (2017). The number of bits necessary is about 25 times smaller (from 1843200 to 72640).

Using the public generator in row echelon form could cause some trouble because the first k coordinates of ciphertexts would be identical to the corresponding plaintext, except possibly for coordinates modified by the error vector. However, Algorithm 8 encodes not the original plaintext but a modified version, as depicted in Step 4.

The `rowScrambler` corresponds to the matrix $\mathbf{S}$ in key generation. However, in practice, the product of Step 3 in Algorithm 7 is not just a random $k \times k$ matrix over $\mathbb{F}_{2^m}$ but a transformation involving this matrix and also the matrices $\mathbf{G}$ and $\mathbf{P}^{-1}$ that reduces the size of the public key in the disk. To see how it works, remember that linear algebra guarantees that the multiplication of a matrix $\mathbf{M}$ by a square full-rank matrix preserves $\mathbf{M}$'s rank. In particular, the product of two invertible matrices is again an invertible matrix. With this in mind, consider the public key, $\mathbf{SGP}^{-1}$, from Algorithm 7, and write it as $\left[\mathbf{A} \mid \mathbf{B}\right]$, where $\mathbf{A}$ is a square matrix of order k and $\mathbf{B}$ is a $k \times (n-k)$ matrix. Since $\mathbf{G}$ has rank k, $\left[\mathbf{A} \mid \mathbf{B}\right]$ also has rank k and can be reduced to row echelon form, i.e., there exists a matrix $\mathbf{T}$, such that $\mathbf{T}\left[\mathbf{A} \mid \mathbf{B}\right] = \left[\mathbf{I} \mid \mathbf{B}'\right]$, and it is easy to see that $\mathbf{T} = \mathbf{A}^{-1}$. Now, if we let $\mathbf{G}' = \mathbf{TSGP}^{-1} = \left[\mathbf{I} \mid \mathbf{B}'\right]$ instead, we can reduce its size by storing only $\mathbf{B}'$ as long as we compensate in the private key, storing $(\mathbf{TS})^{-1} = \mathbf{S}^{-1}\mathbf{T}^{-1} = \mathbf{S}^{-1}\mathbf{A}$. Therefore, in Frame 1, `rowScrambler` contains the encoding of $\mathbf{S}^{-1}\mathbf{A}$, and `PublicKey`'s component `publicGenerator` contains the encoding of $\mathbf{B}'$.

4.3 PKE ALGORITHMS IMPLEMENTATION

This section shows the main pieces of code that implement the three algorithms of our public-key encryption scheme and describes, for each algorithm, how the statements in it map to code explaining the design decisions made. We follow the natural order, first explaining GEN, then ENC, and finally DEC.

4.3.1 GEN Implementation

The keys used in the cryptosystem correspond in code to classes `SecretKey` and `PublicKey`, located in the `key` module. Both store the respective components described in Algorithm 7 (i.e., a `SecretKey` object stores its Gabidulin code and its row and column scrambler matrices). Additionally, the keys store the dimension of the subspace used in key generation. This value is necessary since we must calculate the maximum number of errors to add to the codeword in encryption and check if this number matches the number of errors corrected in decryption. We could have stored this number directly, but as we will see later, we need the subspace dimension to reconstruct the key when importing. The keys also have methods to retrieve the parameters used with keys. Namely, m, n, k, and δ.

The implementation of GEN is divided into two main components. The first is a function that generates a new private key, given the required security level. We implement it as a function because the algorithm does not need to store a state, nor it depends on any object. On the other hand, public keys depend on private keys, and so their generation or, more precisely, their derivation always rely on the previous existence of a private key. Therefore, the second component is a method of the `SecretKey` class.

Listing 1 and Listing 2 depict these two components implementing key generation and, starting in the next paragraph, we describe how the code maps to Algorithm 7 step by step.

```
1  def generate(security_level):
2      m, n, k, delta = _select_parameters(security_level)
3
4      c = _random_gabidulin_code(m, n, k)
5      p = random_invertible_subpace_matrix(GF(2 ** m), delta, n)
6
7      r = random_invertible_matrix(GF(2 ** m), k)
8      t = r * c.generator_matrix() * p.inverse()
9      s = r.inverse() * t.submatrix(ncols=k)
10
11     return SecretKey(c, s, p, delta)
```

Listing 1 – `SecretKey` generation

The first major component of Algorithm 7 implementation is the function `generate` (Listing 1). It takes an integer representing the security level as the parameter and returns a new private key. The first step of GEN corresponds to Line 2 of Listing 1, which takes the security level in variable `security_level` and uses it to call `_select_parameters`, which returns suitable values for k, n, m, and δ, assigning them to the respective variables. The next step of the algorithm maps to Line 4. In this line, the function `_random_gabidulin_code`, with inputs `m`, `n`, and `k` from the previous step, generates a new random (n, k) Gabidulin code over $\mathbb{F}_{2^m}$. We to store the Gabidulin code directly instead of its generator matrix for convenience.

Step 3 of `Gen` generates the row scrambler matrix `s` as a random invertible $k \times k$ matrix. However, in practice, we perform some extra computations to reduce the size of the public keys used in the cryptosystem. Therefore, we skip to Step 4 and revisit Step 3 later. The fourth step produces the column scrambler matrix `p` directly by calling the function `random_invertible_subspace_matrix` from `matrix`. The function takes as input the base field the elements belong to, along with the vector subspace dimension and the matrix order, and returns a matrix with the desired characteristics. Back to Step 3, represented in code by Lines 7, 8, and 8. We first generate a random invertible matrix of order k over $\mathbb{F}_{2^m}$ as in Algorithm 7 and assign it to `r`. Then, we multiply it by the generator matrix of `c` and the inverse of `p`, obtaining `t`. Finally, we compute the matrix `s` by multiplying the inverse of `r` by the submatrix of `t` composed of the first k rows and columns. Frame 2 depicts a sample of a private key in PEM format.

The second major piece of code implementing GEN is the method `public_key` from `SecretKey` (Listing 2). It only takes `self` as a parameter since it is the only dependency and returns its corresponding public key. Line 2 corresponds to Step 5 of Algorithm 7, and it simply takes the three matrices: the row scrambler `_s`, the generator of the secret

```
-----BEGIN PRIVATE KEY-----
MIKDigSCAdBxToQUtnsewc90OnozEAlHTUZ0lDNM+5N+dKaxWwUZbG1woUny/Vi6
smeG0C9aqwnrW+MoBDP6Uz2UqfXKhdhB6tvbV43E3pR9yL3fFmyyYcacbwFEv34d
jYAGCImfPJN6rrB2Y/LF1D9iu8lip7ERodb1QY+moOvzrs4mlZ4F2UbiE8P/1AdL
                                  ⋮
Z3tA+M4gQAC+fPRAsWSPq2d7QPjOIEAAAAAAAAAAAABne0D4ziBAAEfOqUsQjY+r
AAAAAAAAAAC+fPRAsWSPq0fOqUsQjY+rILXps96tz6ueyR3zb8lAAEfOqUsQjY+r
Z3tA+M4gQABne0D4ziBAADAMAgFAAgE6AgEcAgED
-----END PRIVATE KEY-----
```

Frame 2 – PEM-encoded private key sample

code _c, and the column scrambler _p, and multiply them. Note that the resulting matrix is in echelon form.

```
1 def public_key(self):
2     g = self._s.inverse() * self._c.generator_matrix() * self._p.inverse()
3     return PublicKey(g, self._d)
```

Listing 2 – `PublicKey` generation

To finish, the last stop GEN is not implemented in a specific part of the code but is represented by the return statements in Listing 1 and Listing 2. Besides the private and public keys components defined in GEN, the dimension delta of the vector subspace used in generation also composes both. This value is necessary for Steps 1 and 8 of ENC and DEC, respectively. Frame 3 shows an example of a PEM-encoded public key.

```
-----BEGIN PUBLIC KEY-----
MIIaUwOCGkEAa38xq0wSQRbKhGDIeqzOx0LDOfH1euAJaa3X9SrwanwJR3kUsIxC
9RrPLepRmX26M9COsWkkuzOv4mKgZ70AdgNqd5Fz73B2zP+lP/tJ/CfFU6xqp/xo
ANxh7T1l6rvleDYUyV73Uj+spiOqDRjC1bkpKOf7B4ZdpyW123/fMm0M6tZGnwWT
                                  ⋮
H280IBH9bPvg76ihwlLOpWtIVr9+PmLazMeu6FK9k51/o5OiKsz0ioKKGjjEt2ag
eoCiPHaZh6K+hUaPbgV13dBgLw9fg+0BMCialOUDIZqfprlaesfmRUxrz0CWE63n
YmHSnDcTZ3o0MAwCAUACAToCARwCAQM=
-----END PUBLIC KEY-----
```

Frame 3 – PEM-encoded public key sample

4.3.2 ENC Implementation

Contrary to GEN which entry point is a function, ENC, as well as DEC, is a function object or a functor. We implemented them as functors because it more directly reflects the fact that the key to use is fixed. Listing 3 below shows the method `__call__` of class `Enc` that implements the homonym algorithm. Since the object already contains the public key to use, the method `__call__` takes only the bytes of the plaintext we want to encrypt.

```
1  def __call__(self, plaintext):
2      rank = self._decoding_capacity()
3      error = encode(random_rank_vector(self._codeword_space(), rank))
4
5      verifier_hash = self._hash(error + plaintext)
6      extended_plaintext = plaintext + verifier_hash
7
8      error_hash = self._xof(error, len(extended_plaintext))
9
10     message = strxor(extended_plaintext, error_hash)
11     message = decode(message, self._message_space())
12
13     codeword = encode(message * self._key.g())
14     ciphertext = strxor(codeword, error)
15
16     return ciphertext
```

Listing 3 – ENC implementation

Step 1 of Algorithm 8 corresponds to Lines 2 and 3 of Listing 3. The first line computes the value $\frac{n-k}{2\delta}$ from Step 1 by calling the `_decoding_capacity`. As already explained, this value is the maximum number of errors we can add to a codeword in the public code and still decode it using secret code. Then, we use the result, stored in `rank`, and generate a random vector in the codeword space with this rank. Here, we call the method `_codeword_space`, which uses the key parameters to calculate the codeword space, and the function `encode` from module `io` to get the error vector represented as bytes. Next, Line 5 implements Step 2, which concatenates the bytes of the error vector and the plaintext and computes the digest of the result using `_hash` to get the `verifier_hash`, and Line 6 concatenates `plaintext` with `verifier_hash` to obtain the extended plaintext of Step 3.

Step 4 corresponds to Lines 8, 10, and 11. In Line 8, we use the extendable-output function object `_xof` to compute the digest of the error bytes. Since we add this value to the extended plaintext, we know they have to be of the same length so we pass `len(extended_plaintext)` as the second parameter to the method `__call__` of `_xof`. Lines 10 and 11 compute the addition of Step 4 in two parts. The first line calculates the XOR between `extended_plaintext` and `error_hash`, and the second line decodes the resulting bytes into a message vector in the public code. Similar to Line 15, where we called the method `_codeword_space`, we now call `_message_space` to obtain the vector space where the messages in the public code lie and use the returned value as the second parameter to the function `decode`.

Finally, Lines 13, 14, and 16 compute the codeword from the message vector constructed in Line 11, adds the error vector, and returns the encrypted bytes, equating to Step 5 of Algorithm 8. It does by first encoding the `message` with the public code generator and converting the result into bytes in Line 13, and then applying the XOR

operator between the codeword and error bytes in Line 14. In Algorithm 8, this step performs an addition of the error and codeword vectors. However, we have been using the error represented as bytes and the finite field of their elements has characteristic 2, so applying XOR to their bytes is more convenient.

4.3.3 DEC Implementation

As already mentioned, we implement the decryption algorithm as a functor. So, after constructing the `Enc` object passing the private key and the hash and XOF algorithms, one can decrypt ciphertext calling the `__call__` method in Listing 4. It takes the ciphertext as the sole argument and returns either the corresponding plaintext or a `DecodingError`.

```
1  def __call__(self, ciphertext):
2      received_word = decode(ciphertext, self._codeword_space())
3      codeword = self._key.c().decode_to_code(received_word * self._key.p())
4
5      message = encode(self._key.c().unencode(codeword) * self._key.s())
6
7      error_vector = received_word + codeword * self._key.p().inverse()
8      error = encode(error_vector)
9
10     extended_plaintext = strxor(message, self._xof(error, len(message)))
11     plaintext, verifier_hash = self._extract_hash(extended_plaintext)
12
13     hash_verified = verifier_hash == self._hash(error + plaintext)
14     rank_verified = rank_weight(error_vector) == self._decoding_capacity()
15
16     if not hash_verified and not rank_verified:
17         raise DecodingError()
18
19     return plaintext
```

Listing 4 – DEC implementation

The method starts taking the ciphertext bytes and converting them to a vector in the codeword space in Line 2. To this end, it uses the function `decode` from the `io` module. Then, the method implements Steps 1–3 of Algorithm 9 in Line 3, recovering $\mathbf{c}_0$ by multiplying `received_codeword` by the matrix $\mathbf{P}$ and decoding the result with the secret code. In case of a decoding failure, the method `decode_to_code` raises a `DecodingError`, corresponding to Steps 2 and 3 of the algorithm. Otherwise, Line 5 of Listing 4 retrieves $\mathbf{m}_1$ by decoding `codeword` using the method `unencode` from the secret code and multiplying it by the matrix $\mathbf{S}$, also contained in the key. Then, Line 7 computes the error vector $\mathbf{e}$ by multiplying the codeword with the inverse of $\mathbf{P}$ and adding the result to `received_word`, mapping to Step 5 of the algorithm, and Line 8 converts $\mathbf{e}$ to bytes. This conversion is necessary to take its hash value later.

Lines 10 and 11 correspond to Steps 5 and 6 of Algorithm 9. Line 10 computes `extended_plaintext`, which matches $\mathbf{m}_0$ in the algorithm. Again, since we are in a finite field of characteristic 2, the addition of elements corresponds to the XOR of their bytes representation. Then, in Line 11, we get back both the plaintext and the verification hash applying the `_extract_hash` method on `extended_plaintext`. This method uses the expected plaintext length to determine where to split the extended plaintext.

Steps 8 and 9 of Algorithm 9, which checks the verifier hash and rank of the error vector, corresponds to Lines 13–17 of Listing 4. Line 13 computes if the `verifier_hash` obtained before matches the hash calculated from the bytes of the error and the plaintext concatenated, storing the result in `hash_verified`. Line 14 compares the rank of the error vector to $\lfloor \frac{n-k}{2\delta} \rfloor$, obtained through the `_decoding_capacity` function, and stores the result in `rank_verified`. Then, Line 16 checks the variables and, if both are false, it raises a `DecodingError` in Line 17. Otherwise, decryption is successful, and the method returns the plaintext in Line 19.

4.4 USING THE CRYPTOSYSTEM

This section describes the operations one may want to perform using our implementation, such as instantiating a new cipher object, generating and exporting keys, and encrypting and decrypting plaintexts and ciphertext, respectively, considering the situation in which Alice wants to send Bob a message.

4.4.1 Generating a key pair

We start showing how Bob can generate a new key pair, and for this purpose, he needs to choose the desired security level. Three standard security levels used in cryptography are available to him, 128, 192, and 256 bits. Bob opts to use a 128-bits key to receive messages from Alice and creates a new secret key, `sk`, by calling the `generate` function with the value 128 as the argument in Line 1 of Listing 5. Then, he derives the corresponding public key, `pk`, calling the `public_key` method on `sk` in Line 2:

```
1 sk = generate(128)
2 pk = sk.public_key()
```

Listing 5 – Generating a new key pair

4.4.2 Exporting and storing keys

Now that Bob has a key pair, he might want to share the public one with a communicating party, store the private key and send the public to a repository, or even store both keys to use later. In any case, he needs to serialize both keys. Suppose the

communication will not happen immediately, so Bob wants to store `sk` using PEM encoding but send `sk` to Alice using DER. To encode the secret key, Bob calls the method `export_pem` on `sk` as in Line 1 of Listing 6. The method returns a string containing the PEM encoded key that he writes to file "secret.pem" (Lines 3 and 4). Then, he encodes `pk` by calling the method `export_der` (Line 6) and send the resulting bytes to Alice (we deliberately omit the transmission as we are not concerned with the method used for that).

```
1 encoded_sk = sk.export_pem()
2
3 with open("secret.pem", "w") as file:
4     file.write(encoded_sk)
5
6 encoded_pk = pk.export_der()
7
8 # send `encoded_pk` to Alice
```

Listing 6 – Encoding the key pair

4.4.3 Importing a public key and encrypting a message

Now, imagine that on receiving the public key, Alice stores it in a file called "bob_pk.der". Later, when the time comes for Alice to send a message, she imports Bob's public key by opening the file containing the key in DER format, reads its bytes, and calls the function `import_public_der`, passing the bytes as the only argument (Lines 1 and 2 of Listing 7). The function returns a `PublicKey` object representation of the key, which Alice assigns to variable `bob_pk`.

To send a message, Alice needs to create an instance of the `Enc` class. As we discussed before, this requires a public key along with the chosen hash and extendable-output functions. Alice opts to use the `SHA3` and `SHAKE` implementations provided by the hash module and, knowing the key offers 128 bits of security, she instantiates both hash functions passing the value 128 (Line 4). Finally, Alice can use `enc` to encrypt the message she wants to send to Bob and assume it is "the quick brown fox jumps over the lazy dog" (Line 6). Since the plaintext length is less than the input length expected by the `__call__` method of `enc`, it must be padded (for the parameters used in key generation, it expects a 192 bytes input). Alice uses the `pad` function from PyCryptodome to pad the message using PKCS#7 option and encrypts the resulting bytes (Line 7).

```
key_data = open("bob_pk.der", "rb").read()
bob_pk = import_public_der(key_data)

enc = Enc(bob_pk, SHA3(128), SHAKE(128))

plaintext = b"the quick brown fox jumps over the lazy dog"
ciphertext = enc(pad(plaintext, enc.plaintext_len()))

# send Bob the ciphertext
```

Listing 7 – Importing a public key and encrypting a plaintext

4.4.4 Importing a private key and decrypting a ciphertext

On the other side of communication, Bob prepares to receive Alice's message and needs to follow a procedure similar to what Alice did. First, he imports the secret key previously stored in PEM format, opening the file "secret.pem" and reading the string it contains (Line 1 of Listing 8). Bob then imports the key calling the function `import_secret_pem`, passing the read string as an argument, and assigns the returned `SecretKey` to `sk`. With `sk`, he now creates an `Enc` object in Line 4, sending as arguments `sk` and new instances of `SHA3` and `SHAKE`. Again, Bob knows the secret key used provides 128 bits of security and therefore instantiates both the `SHA3` and `SHAKE` accordingly. Bob must use the same algorithms Alice used for the hash and the extendable-output functions in encryption; otherwise, the verification hash computed at one end will not match the verification hash computed at another end. Lastly, Bob decrypts the ciphertext he received using `enc`, and the result should be the original plaintext encrypted by Alice. He undoes the padding made by Alice and retrieves the intended message. Bob used the `unpad` function from PyCryptodome to revert padding, although any piece of code implementing PKCS#7 would suffice.

```
# receive ciphertext from Alice

key_data = open("secret.der", "r").read()
sk = import_secret_pem(key_data)

dec = Dec(sk, SHA3(128), SHAKE(128))

ciphertext = unpad(dec(ciphertext), enc.plaintext_len())
```

Listing 8 – Importing a private key and decrypting a ciphertext

5 FINAL REMARKS

This thesis discussed rank metric cryptography based on Gabidulin codes. We outlined the attacks and corrections these schemes went through, presented an IND-CCA scheme and its implementation, and proposed its parameters based on experiments. We also gave a slight hint of what may come next for this type of cryptography by showing the improvements in attacks in recent years.

The use of this family of codes promised post-quantum cryptographic primitives with key sizes much smaller than other alternatives such as the McEliece cryptosystem but suffered from attacks since its first use. Some of them forced schemes to increase the size of keys, while others completely broke systems. And while new proposals have successfully depleted polynomial-time attacks until now, a fresh understanding of the rank decoding problem (Definition 3.1) allowed the cryptanalysis of this type of scheme to disregard their specific structure. Despite all this pessimistic scenario, NIST recommends the continuation in the studies of rank metric cryptosystems since they offer an attractive alternative to Hamming metric with comparable bandwidth (ALAGIC et al., 2020). We believe the topic needs further research, either in the direction of reducing the key sizes through techniques or in the opposite direction, developing more powerful attacks to discard the use of these schemes in practice.

REFERENCES

ABDOULI, Ameera Salem Al et al. DRANKULA: A McEliece-like Rank Metric based Cryptosystem Implementation. In: SAMARATI, Pierangela; OBAIDAT, Mohammad S. (Eds.). **Proceedings of the 15th International Joint Conference on e-Business and Telecommunications, ICETE 2018 - Volume 2: SECRYPT, Porto, Portugal, July 26-28, 2018**. [S.l.]: SciTePress, 2018. P. 230–241. DOI: 10.5220/0006838102300241. Available from: <https://doi.org/10.5220/0006838102300241>. Cit. on p. 41.

ALAGIC, Gorjan et al. Status report on the second round of the NIST post-quantum cryptography standardization process. **US Department of Commerce, NIST**, 2020. Cit. on p. 59.

ARAGON, Nicolas et al. A New Algorithm for Solving the Rank Syndrome Decoding Problem. In: 2018 IEEE International Symposium on Information Theory (ISIT). [S.l.: s.n.], 2018. P. 2421–2425. DOI: 10.1109/ISIT.2018.8437464. Cit. on p. 43.

BARDET, Magali; BRIAUD, Pierre, et al. An Algebraic Attack on Rank Metric Code-Based Cryptosystems. In: CANTEAUT, Anne; ISHAI, Yuval (Eds.). **Advances in Cryptology – EUROCRYPT 2020**. Cham: Springer International Publishing, 2020. P. 64–93. Cit. on p. 43.

BARDET, Magali; BROS, Maxime, et al. Improvements of Algebraic Attacks for Solving the Rank Decoding and MinRank Problems. In: MORIAI, Shiho; WANG, Huaxiong (Eds.). **Advances in Cryptology – ASIACRYPT 2020**. Cham: Springer International Publishing, 2020. P. 507–536. Cit. on p. 43.

BERLEKAMP, E.; MCELIECE, R.; TILBORG, H. van. On the inherent intractability of certain coding problems (Corresp.) **IEEE Transactions on Information Theory**, v. 24, n. 3, p. 384–386, 1978. DOI: 10.1109/TIT.1978.1055873. Cit. on p. 35.

COGGIA, Daniel; COUVREUR, Alain. On the security of a Loidreau rank metric code based encryption scheme. **Designs, Codes and Cryptography**, Springer Verlag, v. 88, n. 9, p. 1941–1957, Sept. 2020. Long version of an article accepted at the conference WCC 2019. DOI: 10.1007/s10623-020-00781-4. Available from: <https://hal.archives-ouvertes.fr/hal-03049694>. Cit. on p. 39.

DELSARTE, Ph. Bilinear forms over a finite field, with applications to coding theory. **Journal of Combinatorial Theory, Series A**, v. 25, n. 3, p. 226–241, 1978. ISSN

0097-3165. DOI: https://doi.org/10.1016/0097-3165(78)90015-8. Available from: <https://www.sciencedirect.com/science/article/pii/0097316578900158>. Cit. on p. 30.

FUJISAKI, Eiichiro; OKAMOTO, Tatsuaki. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In: PROCEEDINGS of the 19th Annual International Cryptology Conference on Advances in Cryptology. Berlin, Heidelberg: Springer-Verlag, 1999. (CRYPTO '99), p. 537–554. Cit. on p. 40.

FUJISAKI, Eiichiro; OKAMOTO, Tatsuaki. Secure Integration of Asymmetric and Symmetric Encryption Schemes. **J. Cryptol.**, Springer-Verlag, Berlin, Heidelberg, v. 26, n. 1, p. 80–101, Jan. 2013. ISSN 0933-2790. DOI: 10.1007/s00145-011-9114-1. Available from: <https://doi.org/10.1007/s00145-011-9114-1>. Cit. on p. 40.

GABIDULIN, E. M.; PARAMONOV, A. V.; TRETJAKOV, O. V. Ideals over a Non-Commutative Ring and their Application in Cryptology. In: DAVIES, Donald W. (Ed.). **Advances in Cryptology — EUROCRYPT '91**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991. P. 482–489. Cit. on pp. 15, 35, 36.

GABIDULIN, E.M.; OURIVSKI, A.V. Modified GPT PKC with Right Scrambler. **Electronic Notes in Discrete Mathematics**, v. 6, p. 168–177, 2001. WCC2001, International Workshop on Coding and Cryptography. ISSN 1571-0653. DOI: https://doi.org/10.1016/S1571-0653(04)00168-4. Available from: <https://www.sciencedirect.com/science/article/pii/S1571065304001684>. Cit. on p. 37.

GABIDULIN, E.M.; OURIVSKI, A.V., et al. Reducible rank codes and their applications to cryptography. **IEEE Transactions on Information Theory**, v. 49, n. 12, p. 3289–3293, Dec. 2003. ISSN 1557-9654. DOI: 10.1109/TIT.2003.820038. Cit. on p. 37.

GABIDULIN, Ernst M. Attacks and counter-attacks on the GPT public key cryptosystem. **Des. Codes Cryptogr.**, v. 48, n. 2, p. 171–177, 2008. DOI: 10.1007/s10623-007-9160-8. Available from: <https://doi.org/10.1007/s10623-007-9160-8>. Cit. on p. 38.

GABIDULIN, Ernst M. Theory of Codes with Maximum Rank Distance. **Probl. Peredachi Inf.**, v. 21, p. 3–16, 1 1985. Cit. on p. 30.

GABIDULIN, Ernst M.; RASHWAN, Haitham; HONARY, Bahram. On Improving Security of GPT Cryptosystems. In: PROCEEDINGS of the 2009 IEEE International Conference on Symposium on Information Theory - Volume 2. Coex, Seoul, Korea: IEEE Press, 2009. (ISIT'09), p. 1110–1114. Cit. on p. 38.

GABORIT, Philippe; RUATTA, Olivier; SCHREK, Julien. On the Complexity of the Rank Syndrome Decoding Problem. **IEEE Transactions on Information Theory**, v. 62, n. 2, p. 1006–1019, 2016. DOI: 10.1109/TIT.2015.2511786. Cit. on pp. 36, 43.

GABORIT, Philippe; ZÉMOR, Gilles. On the Hardness of the Decoding and the Minimum Distance Problems for Rank Codes. **IEEE Transactions on Information Theory**, v. 62, n. 12, p. 7245–7252, 2016. DOI: 10.1109/TIT.2016.2616127. Cit. on p. 35.

GHATAK, Anirban. Extending Coggia–Couvreur attack on Loidreau's rank-metric cryptosystem. **Designs, Codes and Cryptography**, v. 90, p. 215–238, 1 2022. ISSN 1573-7586. DOI: 10.1007/s10623-021-00972-7. Available from: <https://doi.org/10.1007/s10623-021-00972-7>. Cit. on p. 39.

GIBSON, Keith. Severely Denting the Gabidulin Version of the McEliece Public Key Cryptosystem. **Des. Codes Cryptogr.**, v. 6, n. 1, p. 37–45, 1995. DOI: 10.1007/BF01390769. Available from: <https://doi.org/10.1007/BF01390769>. Cit. on p. 37.

GIBSON, Keith. The Security of the Gabidulin Public Key Cryptosystem. In: PROCEEDINGS of the 15th Annual International Conference on Theory and Application of Cryptographic Techniques. Saragossa, Spain: Springer-Verlag, 1996. (EUROCRYPT'96), p. 212–223. Cit. on p. 37.

GROVER, Lov K. A Fast Quantum Mechanical Algorithm for Database Search. In: PROCEEDINGS of the Twenty-Eighth Annual ACM Symposium on Theory of Computing. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996. (STOC '96), p. 212–219. DOI: 10.1145/237814.237866. Available from: <https://doi.org/10.1145/237814.237866>. Cit. on p. 14.

HOFHEINZ, Dennis; HÖVELMANNS, Kathrin; KILTZ, Eike. A Modular Analysis of the Fujisaki-Okamoto Transformation. In: KALAI, Yael; REYZIN, Leonid (Eds.). **Theory of Cryptography**. Cham: Springer International Publishing, 2017. P. 341–371. Cit. on p. 40.

LIDL, Rudolf; NIEDERREITER, Harald. **Finite Fields**. 2. ed. [S.l.]: Cambridge University Press, 1996. (Encyclopedia of Mathematics and its Applications). DOI: 10.1017/CBO9780511525926. Cit. on p. 23.

LOIDREAU, Pierre. A New Rank Metric Codes Based Encryption Scheme. In: LANGE, Tanja; TAKAGI, Tsuyoshi (Eds.). **Post-Quantum Cryptography**. Cham: Springer International Publishing, 2017. P. 3–17. Cit. on pp. 38, 40, 50.

LOIDREAU, Pierre; PHAM, Ba-Duc. An analysis of Coggia-Couvreur Attack on Loidreau's Rank-metric public-key encryption scheme in the general case. working paper or preprint. [S.l.], Jan. 2022. Available from: <https://hal.archives-ouvertes.fr/hal-03514106>. Cit. on p. 39.

MCELIECE, R. J. A Public-Key Cryptosystem Based On Algebraic Coding Theory. **Deep Space Network Progress Report**, v. 44, p. 114–116, Jan. 1978. Cit. on p. 35.

MULLEN, Gary L; PANARIO, Daniel. **Handbook of Finite Fields**. [S.l.]: CRC Press, 2013. Cit. on p. 23.

OTMANI, Ayoub; KALACHI, Hervé Talé; NDJEYA, Sélestin. Improved cryptanalysis of rank metric schemes based on Gabidulin codes. **Des. Codes Cryptogr.**, v. 86, n. 9, p. 1983–1996, 2018. DOI: 10.1007/s10623-017-0434-5. Available from: <https://doi.org/10.1007/s10623-017-0434-5>. Cit. on p. 38.

OURIVSKI, A. V.; JOHANSSON, T. New Technique for Decoding Codes in the Rank Metric and Its Cryptography Applications. In: p. 237–246. DOI: 10.1023/A:1020369320078. Available from: <https://doi.org/10.1023/A:102036932007>. Cit. on p. 43.

OVERBECK, Raphael. A New Structural Attack for GPT and Variants. In: DAWSON, Ed; VAUDENAY, Serge (Eds.). **Progress in Cryptology - Mycrypt 2005, First International Conference on Cryptology in Malaysia, Kuala Lumpur, Malaysia, September 28-30, 2005, Proceedings**. [S.l.]: Springer, 2005. (Lecture Notes in Computer Science), p. 50–63. DOI: 10.1007/11554868_5. Available from: <https://doi.org/10.1007/11554868%5C_5>. Cit. on pp. 37, 38.

OVERBECK, Raphael. Structural attacks for public key cryptosystems based on Gabidulin codes. **Journal of cryptology**, Springer, v. 21, n. 2, p. 280–301, 2008. Cit. on pp. 37, 38.

RASHWAN, Haitham; GABIDULIN, Ernst M.; HONARY, Bahram. Security of the GPT cryptosystem and its applications to cryptography. **Security and Communication Networks**, v. 4, n. 8, p. 937–946, 2011. DOI: https://doi.org/10.1002/sec.228. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sec.228. Available from: <https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.228>. Cit. on p. 38.

ROTH, R.M. Maximum-rank array codes and their application to crisscross error correction. **IEEE Transactions on Information Theory**, v. 37, n. 2, p. 328–336, 1991. DOI: 10.1109/18.75248. Cit. on p. 30.

SAITO, Tsunekazu; XAGAWA, Keita; YAMAKAWA, Takashi. Tightly-Secure Key-Encapsulation Mechanism in the Quantum Random Oracle Model. In: NIELSEN, Jesper Buus; RIJMEN, Vincent (Eds.). **Advances in Cryptology – EUROCRYPT 2018**. Cham: Springer International Publishing, 2018. P. 520–551. Cit. on p. 40.

SHEHHI, Hamad Al et al. An IND-CCA-Secure Code-Based Encryption Scheme Using Rank Metric. In: BUCHMANN, Johannes; NITAJ, Abderrahmane; RACHIDI, Tajjeeddine (Eds.). **Progress in Cryptology – AFRICACRYPT 2019**. Cham: Springer International Publishing, 2019. P. 79–96. Cit. on pp. 15, 40, 41, 48, 66.

SHOR, Peter W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. **SIAM Journal on Computing**, v. 26, n. 5, p. 1484–1509, 1997. DOI: 10.1137/S0097539795293172. eprint: https://doi.org/10.1137/S0097539795293172. Available from: <https://doi.org/10.1137/S0097539795293172>. Cit. on p. 14.

Annex

ANNEX A – SBC FORMAT ARTICLE

A.1 AN IND-CCA RANK METRIC ENCRYPTION SCHEME IMPLEMENTATION

Abstract: The advances in quantum computation impose a severe threat to the public-key cryptographic primitives used nowadays and will turn them completely obsolete if these computers are produced in large scale. Post-quantum cryptography is the area of research that aims to develop cryptographic systems to resist against these attacks. This work considers the use of Gabidulin codes in the construction of encryption schemes. We show that, while providing the possibility of small key sizes, they are especially challenging in terms of security. Then, we present an IND-CCA scheme, proposed by (SHEHHI et al., 2019), provide an implementation, and analyze the theoretical complexity of recent attacks with respect to the scheme and propose a set of parameters to use.

An IND-CCA Rank Metric Encryption Scheme Implementation

Filipe O. de Borba[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brazil

filipe.borba@grad.ufsc.br

Abstract. *The advances in quantum computation impose a severe threat to the public-key cryptographic primitives used nowadays and will turn them completely obsolete if these computers are produced in large scale. Post-quantum cryptography is the area of research that aims to develop cryptographic systems to resist against these attacks. This work considers the use of Gabidulin codes in the construction of encryption schemes. We show that, while providing the possibility of small key sizes, they are especially challenging in terms of security. Then, we present an IND-CCA scheme, proposed by [Shehhi et al. 2019], provide an implementation, and analyze the theoretical complexity of recent attacks with respect to the scheme and propose a set of parameters to use.*

1. Introduction

Our world runs on software. From sensor networks to online banking, from social networks to electronic voting. These applications shape our economy, society, and the way we live like never seen before. Nonetheless, to be useful, these systems need to exchange information. More important, this communication must happen in a manner that unauthorized parts do not participate in the sense that information transmitted is not disclosed to them for as long as its secrecy is necessary.

Cryptography provides the basic building blocks which cryptographic functionalities develop on top to secure such systems. Three of the main cryptographic functionalities used nowadays are public-key encryption, digital signatures, and key exchange, and rely on well-known number theoretical problems considered to have no efficient solution on a classical computer. However, it turns out not to be true for quantum computers, and a remarkable result due to [Shor 1997] renders public-key cryptography completely useless by solving these problems mentioned in subexponential time using a quantum computer.

Taking into account these facts, an international community composed by academia, industry, and government organizations, is working on the task of developing, testing, and standardizing new quantum-resistant primitives. To this new field of study is given the name of post-quantum cryptography in allusion to the scenery after the deployment of quantum computers, although for the quantum skeptic, this is a misnomer to the quest. Among these initiatives, the most prominent is that of the National Institute of Standards and Technology of the United States. The agency initiated a process to standardize post-quantum cryptographic primitives that offer resistance to both classical and quantum computers. Besides that, these primitives should interoperate with current systems and protocols. The role of NIST in the standardization of cryptography is well-known due to its Advanced Encryption Standard competition that selected Rijndael

as the encryption algorithm to be used by the U.S Government and, voluntarily, by the private sector. Currently, the process is in its third round. Among the main families of proposed post-quantum primitives are hash based signatures, lattice-based cryptography, multivariate polynomial cryptography, and code-based cryptography.

This work concerns a specific branch of the latter which relies on rank metric codes. The objective is to introduce post-quantum cryptography using rank metric codes through the example of a cryptosystem, proposed by [Shehhi et al. 2019], and an implementation of it. Along the way we give an overview of this field of research, showing the origins of rank metric cryptography with the work in [Gabidulin et al. 1991], the cat-and-mouse game played by variants of this scheme and attacks breaking them, and a proposal that rules out polynomial-time attacks, and discuss how recent attacks impacts on the key sizes for the scheme we implement.

2. Preliminaries

A finite field is an algebraic structure composed of a non-empty set S equipped two binary operations called addition and multiplication, closed on S. Both operations are associative and have uniquely determined identity and inverse elements in S. Moreover, it holds the distributive property of multiplication over addition can multiplication. A finite field has always p^n elements, where p is a prime and n is a positive integer. Any two finite fields with the same number of elements are isomorphic. We use $\mathbb{F}_{q^n}$ when representing the finite field with q^n elements as polynomials of degree less than n and $\mathbb{F}_q^n$ when representing it as an n-dimensional vector space over $\mathbb{F}_q$.

A (n, k) linear code over $\mathbb{F}_{q^m}$ is defined as a k-dimensional subspace $\mathcal{C}$ of $\mathbb{F}_{q^m}^n$. A code has an associate distance measure function and in rank metric the distance between two elements is given by the rank of the difference between their matrix representations given by the mapping defined below.

Definition 2.1 (Subfield mapping)**.** Let $B = \beta_1, \ldots, \beta_m$ be a basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. The subfield mapping of a vector $\mathbf{v} = \begin{pmatrix} \alpha_1 & \ldots & \alpha_n \end{pmatrix}$ in $\mathbb{F}_{q^m}^n$ with respect to B is given by the map $f_B(\mathbf{v}) = (a_{ij}) \in \mathbb{F}_q^{m \times n}$ such that

$$\alpha_i = \sum_{j=1}^{m} a_{ij}\beta_j, \quad 1 \leq i \leq n.$$

Given that, one can compute the distance between any two vectors in a rank metric code $\mathcal{C}$ as in Definition 2.2. For any vector $\mathbf{v}$, its rank weight is measured as the distance between $\mathbf{v}$ and the zero vector.

Definition 2.2 (Rank distance)**.** Let f be the subfield mapping in Definition 2.1. The rank distance between any two elements $\mathbf{u}, \mathbf{v} \in \mathbb{F}_{q^m}^n$ is given by

$$\delta(\mathbf{u}, \mathbf{v}) = \mathrm{rank}(f_B(\mathbf{u}) - f_B(\mathbf{v})).$$

Gabidulin codes are an especial class of error-correcting codes using rank distance instead of the more widely known Hamming distance. These codes were first introduced

in [Delsarte 1978]. Later, [Gabidulin 1985] proved several properties and showed an efficient decoding algorithm. Given the significance of these contributions, this class of codes receives its name after the author. [Roth 1991] also discovered this class of codes, independently of previous works. An important concept necessary to define Gabidulin codes is that of linearized polynomial (Definition 2.3).

Definition 2.3 (Linearized polynomial)**.** A polynomial $p(x)$ is a linearized polynomial if it has the form

$$p(x) = \sum_{i=0}^{d} a_i x^{q^i}, \quad a_i \in \mathbb{F}_{q^m},\ \forall i \in [0, d]. \tag{1}$$

We denote the set of univariate linearized polynomials over $\mathbb{F}_{q^m}$ with indeterminate x by $\mathbb{L}_{q^m}$. They have the important property that for all $p(x)$ in $\mathbb{L}_{q^m}$, all α, β in $\mathbb{F}_{q^m}$, and all c in $\mathbb{F}_q$, it holds that $p(\alpha + \beta) = p(\alpha) + p(\beta)$, and $p(c\alpha) = cp(\alpha)$.

Definition 2.4 (Gabidulin code)**.** Let $1 \leq k < n \leq m$ be integers, and $g_1, \ldots, g_n \in \mathbb{F}_{q^m}$ be linearly independent elements over $\mathbb{F}_q$. An (n, k) Gabidulin code over $\mathbb{F}_{q^m}$ defined at points $g_1, \ldots, g_n$ is the set of code words, each of which is defined as $\begin{pmatrix} p(g_1) & \cdots & p(g_n) \end{pmatrix}$, for a distinct linearized polynomial p over $\mathbb{F}_{q^m}$ of degree less than q^k.

In practice, each possible polynomial p corresponds to a different message in the message space. These codes can correct errors at a rank distance up to $\lfloor \frac{n-k}{2} \rfloor$ in polynomial time.

Definition 2.5 (Generator matrix of a Gabidulin code)**.** Let $\mathcal{C}$ be an (n, k) Gabidulin code defined over points $g_0, \ldots, g_{n-1} \in \mathbb{F}_{q^m}$. Then, the generator matrix $\mathbf{G}$ of $\mathcal{C}$ has the following form

$$\mathbf{G} = \begin{pmatrix} g_1 & g_2 & \cdots & g_n \\ g_1^q & g_2^q & \cdots & g_n^q \\ \vdots & \vdots & \ddots & \vdots \\ g_1^{q^{k-1}} & g_2^{q^{k-1}} & \cdots & g_n^{q^{k-1}} \end{pmatrix}. \tag{2}$$

We call the vector $\mathbf{g} = \begin{pmatrix} g_1 & g_2 & \cdots & g_n \end{pmatrix}$, composed of the elements from the first row of $\mathbf{G}$, the generator vector of $\mathcal{C}$. It is not unique—nor the generator matrix—as any vector $\alpha\mathbf{g}$, with $\alpha \in \mathbb{F}_{q^m}$, is also a generator vector of $\mathcal{C}$.

Lemma 2.1 (Check matrix form of a Gabidulin code)**.** *Let $\mathcal{C}$ be an (n, k) Gabidulin code over $\mathbb{F}_{q^m}$ defined at points $g_1, g_2, \ldots, g_n$ and let $h_1, h_2, \ldots, h_n \in \mathbb{F}_{q^m}$ such that $\sum_{i=1}^{n} g_i^{q^j} h_i = 0$ for $j \in [k - n + 1, k - 1]$. The parity-check matrix $\mathbf{H}$ of $\mathcal{C}$ is defined as*

$$\mathbf{H} = \begin{pmatrix} h_1 & h_2 & \cdots & h_n \\ h_1^q & h_2^q & \cdots & h_n^q \\ \vdots & \vdots & \ddots & \vdots \\ h_1^{q^{n-k-1}} & h_2^{q^{n-k-1}} & \cdots & h_n^{q^{n-k-1}} \end{pmatrix} \tag{3}$$

That is, the values h_i are a solution for the system of linear equations involving the points at which generated the code. In practice, if we let $\mathbf{G}$ be a generator matrix of $\mathcal{C}$, each of these equations corresponds to the calculation of an entry of the matrix $\mathbf{G}\mathbf{H}^\top$. Since the system if homogeneous, the matrix $\mathbf{H}$ is the parity-check matrix of $\mathcal{C}$. To ilustrate, take for instance, $j = k - n + 1$, obtaining the equation.

$$\sum_{i=1}^{n} g_i^{q^{k-n+1}} h_i = \sum_{i=1}^{n} \left(g_i^{q^{k-n+1}} h_i \right)^{q^{n-k-1}} = \sum_{i=1}^{n} g_i h_i^{q^{n-k-1}} = 0 \tag{4}$$

This corresponds exactly to the last element in the first row of $\mathbf{GH}^\top$. Similarly, other values of j results in the equation for different entries of $\mathbf{GH}^\top$.

3. Rank Metric Cryptography

The first public-key encryption scheme lying on the theory of error-correcting codes appeared in [McEliece 1978]. In this work, the author proposes an encryption/decryption scheme analogous to encoding/decoding but, to prevent unauthorized parties from accessing the messages, the scheme hides the generator matrix $\mathbf{G}$, by computing a matrix $\mathbf{G}' = \mathbf{SGP}$, and encrypts messages by encoding them with $\mathbf{G}'$ and adding a certain number of errors to the resulting codeword. The idea is that if $\mathbf{G}'$ generates a random looking linear code, then the ciphertext should look like a codeword from a random linear code, for which decoding is NP-complete [Berlekamp et al. 1978], and only who has $\mathbf{S}$, $\mathbf{G}$, and $\mathbf{P}$ can recover the message. The main disadvantage of McEliece's proposal is the size of the keys that is prohibitive for many applications.

An alternative offering smaller key sizes came with [Gabidulin et al. 1991] proposing the GPT, in honor of its authors, and adapted McEliece's ideas to Gabidulin codes. It relies on the rank decoding problem, presented in Definition 3.1, and the argument is that decoding a codeword from a random code in rank metric is exponentially more difficult than decoding one from a random code in Hamming metric [Gaborit et al. 2016]. Even if the rank decoding problem is not known to be NP-complete, there is a randomized reduction to an NP-complete problem [Gaborit and Zémor 2016]. Next, we introduce the GPT cryptosystem.

Definition 3.1 (Rank decoding problem)**.** Let $\mathbf{G} \in \mathbb{F}_{q^m}^{k \times n}$ be a matrix, $\mathbf{c} \in \mathbb{F}_{q^m}^{n}$ be a vector, and w an integer. Find $\mathbf{m} \in \mathbb{F}_{q^m}^{k}$ and $\mathbf{e} \in \mathbb{F}_{q^m}^{n}$ such that $\mathbf{c} = \mathbf{mG} + \mathbf{e}$ and the rank weight of $\mathbf{e}$ is less than or equal w?

The GPT system suffered its first attack in [Gibson 1995] and [Gibson 1996], where the cryptanalytic algorithm presented runs in exponential time and only worked for the small parameters proposed at that time. Latter works introduced variants of GPT that attempted to better hide the structure of the secret code. In [Gabidulin and Ourivski 2001] and [Gabidulin et al. 2003] the authors returned to the idea of using a column scrambler matrix $\mathbf{P}$ as in the original McEliece proposal to avoid Gibson's attack. However, [Overbeck 2005] and [Overbeck 2008] completely broke the system using the strong structure of Gabidulin codes and the fact $\mathbf{P}$ is defined over $\mathbb{F}_q$. Even subsequent schemes claiming security against the Overbeck attack by taking the elements from $\mathbb{F}_{q^m}$ instead of $\mathbb{F}_{q^m}$, such as [Gabidulin 2008], [Gabidulin et al. 2009], and [Rashwan et al. 2011], could not stand. [Otmani et al. 2018] have shown latter that is possible to reformulate all of them as instances of the system attacked by Overbeck. The solution to this problem came only in [Loidreau 2017] which mixed the ideas usually applied in McEliece-like cryptosystems to others ideas arising from the design of low rank parity-check codes (LRPC).

3.1. The Shehhi et al. Cryptosystem

The encryption scheme proposed in [Loidreau 2017] does not achieve indistinguishability under chosen-plaintext attack, which is the minimum security guarantee required

the NIST standardization process. Therefore, it does not provide security against chosen-ciphertext attack as well. Given that, [Shehhi et al. 2019] proposed an IND-CCA variant of Loidreau's cryptosystem borrowing ideas presented in works such as [Fujisaki and Okamoto 1999], [Fujisaki and Okamoto 2013], [Hofheinz et al. 2017], and [Saito et al. 2018], usually applied to a public key encryption scheme to turn it into an IND-CCA key encapsulation mechanism or hybrid encryption scheme. The product of the transformation presented by the authors is nonetheless a PKE scheme.

The new construction, presented in Definition 3.2, demands the use of two hash functions. One is a traditional hash function, referred to as H_0. The other, H_1, is a hash function where one can choose the output size via an extra argument, called an extendable-output function (XOF). It is important to note these functions need to offer at least the same level of security expected for the PKE as a whole. Otherwise its security is reduced to the hash algorithms. We define the Shehhi et al. cryptosystem below.

Definition 3.2 (Shehhi et al. Cryptosystem)**.** The Shehhi et al. cryptosystem is composed by a triple of probabilistic polynomial-time algorithms, (GEN, ENC, DEC), where:

$\textsc{Gen}(1^N)$
1 $k, m, n, \delta := \textsc{SelectParameters}(1^N)$
2 $\mathbf{G} \leftarrow \{\mathbf{M} \in \mathbb{F}_{q^m}^{k \times n} \ : \ \langle \mathbf{M} \rangle \in \mathrm{Gab}(n, k)\}$
3 $\mathbf{S} \leftarrow \{\mathbf{M} \in GL_k(q^m)\}$
4 $\mathbf{P} \leftarrow \{\mathbf{M} \in GL_n(V) \ : \ V \subset \mathbb{F}_{q^m}$ **and** $\dim(V) = \delta\}$
5 $\mathbf{G}' := \mathbf{S}\mathbf{G}\mathbf{P}^{-1}$
6 **return** $pk = \mathbf{G}', \ sk = (\mathbf{G}, \mathbf{S}^{-1}, \mathbf{P})$

Algorithm 1: Shehhi et al. key generation

$\textsc{Enc}_{pk}(\mathbf{m})$
1 $\mathbf{e} \leftarrow \left\{\mathbf{v} \in \mathbb{F}_{q^m}^n : \omega(\mathbf{v}) = \left\lfloor \frac{n-k}{2\delta} \right\rfloor\right\}$
2 $\mathbf{v} := \mathrm{H}_0(\mathbf{e}, \mathbf{m})$
3 $\mathbf{m}_0 := (\mathbf{m} \quad \mathbf{v})$
4 $\mathbf{m}_1 := \mathbf{m}_0 + \mathrm{H}_1(\mathbf{e})$
5 **return** $\mathbf{m}_1\mathbf{G}' + \mathbf{e}$

Algorithm 2: Shehhi et al. encryption

$\textsc{Dec}_{sk}(\mathbf{c})$
1 $\mathbf{c}_0 := \textsc{Decode}(\mathbf{c}\mathbf{P}, \mathbf{G})$
2 **if** $\mathbf{c}_0 = \perp$
3 **return** $\perp$
4 $\mathbf{m}'_1 := \mathbf{c}_0\mathbf{S}^{-1}$
5 $\mathbf{e}' := \mathbf{c} - \mathbf{c}_0\mathbf{P}^{-1}$
6 $\mathbf{m}'_0 := \mathbf{m}'_1 - \mathrm{H}_1(\mathbf{e}')$
7 $(\mathbf{m}' \quad \mathbf{v}') := \mathbf{m}'_0$
8 **if** $\mathrm{H}_0(\mathbf{e}', \mathbf{m}') \neq \mathbf{v}'$ **and** $\omega(\mathbf{e}') \neq \left\lfloor \frac{n-k}{2\delta} \right\rfloor$
9 **return** $\perp$
10 **return** $\mathbf{m}'$

Algorithm 3: Shehhi et al. decryption

The transformation proposed by the authors allows one to encrypt an amount of data larger than the Fujisaki-Okamoto's at a cost of 23% in decryption only. It can be employed, for example, as KEM to exchange multiple keys in one ciphertext. The security proof in [Shehhi et al. 2019] relies on properties specific to the Loidreau's scheme. Nevertheless, the authors claim that the transformation might be adapted to other schemes as well.

4. Analysis of the Security of Rank Based Cryptosystems

Now we analyze rank-based encryption schemes concerning their security, key sizes, and code rates. To this end, we compute the theoretical complexity of recent attacks to our cryptosystem and depict how these attacks have affected the choice of parameters over the last years. We also compare the best attack using different techniques against rank-based systems.

There are two possible forms of breaking a scheme based on rank metric codes. The first is to recover a decoder for the underlying code, the class to which the Overbeck attack mentioned earlier belongs. We disregard this attack as recent proposals seem to hinder this menace. However, for the Loidreau cryptosystem specifically, the Coggia and Covreur attack and its extensions restrict the choice of the subspace dimension δ; the designer has to make a trade-off between the private key size and the amount of redundant information in each ciphertext. Another form is to recover the message from the ciphertext via a combinatorial or algebraic approach.

Combinatorial attacks have been studied for longer, with [Aragon et al. 2018] and [Gaborit et al. 2016] the most remarkable works and although these attacks can make use of Grover's algorithm to speed up, they are not as dangerous as the recent algebraic attacks. [Gaborit et al. 2016, Bardet et al. 2020a, Bardet et al. 2020b] accelerated an area in which relevant discoveries did not show since [Ourivski and Johansson 2002]. Even if these attacks solve the rank decoding problem in exponential time, the advance was enough to left schemes such as ROLLO and RQC out of NIST's third round.

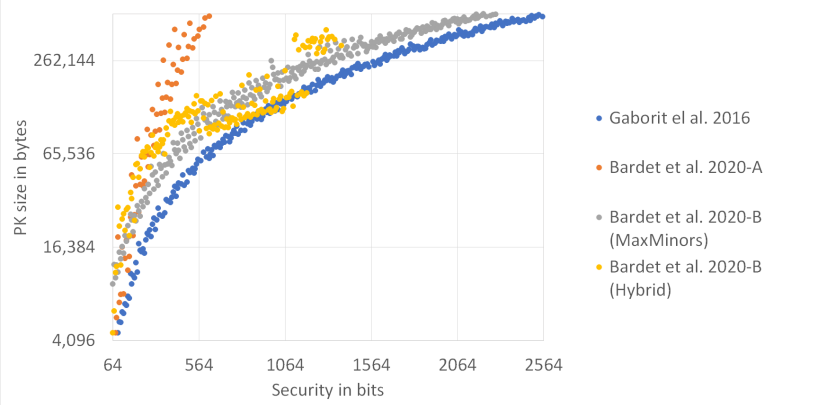


Figure 1. Minimum key sizes to resist algebraic attacks

From Figure 1, it is easy to note that the attack in [Bardet et al. 2020a] needs much larger public-key sizes than the other as the security increases. However, Figure 2 focuses

on the security levels relevant to use in practice, and it is possible to observe a different scenario. For them, the two attacks in [Bardet et al. 2020b] force the scheme to have larger sizes in their public keys than the one in [Bardet et al. 2020a].

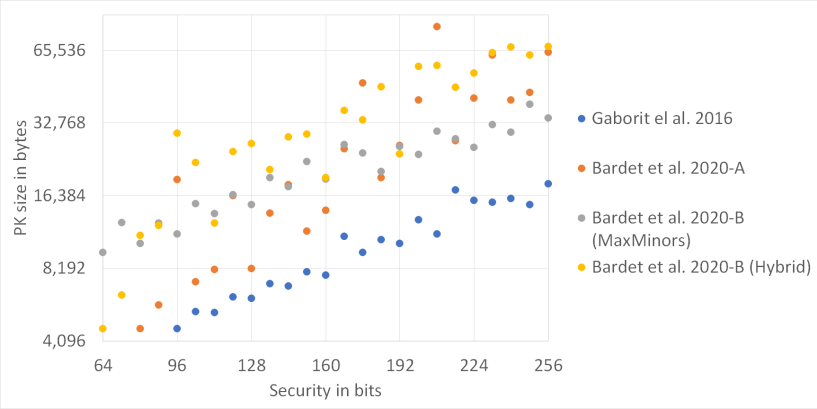


Figure 2. Key sizes of Figure 1 for practical security levels

A feature noticed through our experiments is that a significant variation occurs in the sizes of keys offering the same level of security, even when considering a single attack. As we can see in the graph of Figure 2, on the line of 96 bits of security, there is a set of parameters whose public key must have about 30 KB to offer approximately that security against hybrid attack in [Bardet et al. 2020b]. Nevertheless, it is possible to choose parameters that n way that the resulting key is shorter and more secure, such as the one in the 160-bit line, with approximately 25 KB. This observation highlights the importance of carefully choosing the parameters of our system.

We finish defining the parameters used in our implementation (available at github.com/flpborba/shehhi).Table 1 shows the parameters m, n, k, and δ, and the public and private keys sizes, for the more widely used security levels.

Table 1. Selected parameters.

Security	m	n	k	δ	r	**PK size**	**SK size**
80	72	68	38	3	5	10260	15369
128	80	80	44	3	6	15840	22590
192	104	100	50	3	8	32500	37589
256	168	116	60	3	9	70560	83145

5. Conclusion

This thesis discussed rank metric cryptography based on Gabidulin codes. We outlined the attacks and corrections these schemes went through, presented an IND-CCA scheme and its implementation, and proposed its parameters based on experiments. We also gave a slight hint of what may come next for this type of cryptography by showing the improvements in attacks in recent years. Despite all this pessimistic scenario, NIST recommends

the continuation in the studies of rank metric cryptosystems since they offer an attractive alternative to Hamming metric with comparable bandwidth [Alagic et al. 2020]. We believe the topic needs further research, either in the direction of reducing the key sizes through techniques or in the opposite direction, developing more powerful attacks to discard the use of these schemes in practice.

References

Alagic, G., Alperin-Sheriff, J., Apon, D., Cooper, D., Dang, Q., Kelsey, J., Liu, Y.-K., Miller, C., Moody, D., Peralta, R., et al. (2020). Status report on the second round of the nist post-quantum cryptography standardization process. *US Department of Commerce, NIST*.

Aragon, N., Gaborit, P., Hauteville, A., and Tillich, J.-P. (2018). A new algorithm for solving the rank syndrome decoding problem. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 2421–2425.

Bardet, M., Briaud, P., Bros, M., Gaborit, P., Neiger, V., Ruatta, O., and Tillich, J.-P. (2020a). An algebraic attack on rank metric code-based cryptosystems. In Canteaut, A. and Ishai, Y., editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 64–93, Cham. Springer International Publishing.

Bardet, M., Bros, M., Cabarcas, D., Gaborit, P., Perlner, R., Smith-Tone, D., Tillich, J.-P., and Verbel, J. (2020b). Improvements of algebraic attacks for solving the rank decoding and minrank problems. In Moriai, S. and Wang, H., editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 507–536, Cham. Springer International Publishing.

Berlekamp, E., McEliece, R., and van Tilborg, H. (1978). On the inherent intractability of certain coding problems (corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386.

Delsarte, P. (1978). Bilinear forms over a finite field, with applications to coding theory. *Journal of Combinatorial Theory, Series A*, 25(3):226–241.

Fujisaki, E. and Okamoto, T. (1999). Secure integration of asymmetric and symmetric encryption schemes. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, page 537–554, Berlin, Heidelberg. Springer-Verlag.

Fujisaki, E. and Okamoto, T. (2013). Secure integration of asymmetric and symmetric encryption schemes. *J. Cryptol.*, 26(1):80–101.

Gabidulin, E. and Ourivski, A. (2001). Modified gpt pkc with right scrambler. *Electronic Notes in Discrete Mathematics*, 6:168–177. WCC2001, International Workshop on Coding and Cryptography.

Gabidulin, E., Ourivski, A., Honary, B., and Ammar, B. (2003). Reducible rank codes and their applications to cryptography. *IEEE Transactions on Information Theory*, 49(12):3289–3293.

Gabidulin, E. M. (1985). Theory of codes with maximum rank distance. *Probl. Peredachi Inf.*, 21:3–16.

Gabidulin, E. M. (2008). Attacks and counter-attacks on the gpt public key cryptosystem. *Des. Codes Cryptogr.*, 48(2):171–177.

Gabidulin, E. M., Paramonov, A. V., and Tretjakov, O. V. (1991). Ideals over a non-commutative ring and their application in cryptology. In Davies, D. W., editor, *Advances in Cryptology — EUROCRYPT '91*, pages 482–489, Berlin, Heidelberg. Springer Berlin Heidelberg.

Gabidulin, E. M., Rashwan, H., and Honary, B. (2009). On improving security of gpt cryptosystems. In *Proceedings of the 2009 IEEE International Conference on Symposium on Information Theory - Volume 2*, ISIT'09, page 1110–1114. IEEE Press.

Gaborit, P., Ruatta, O., and Schrek, J. (2016). On the complexity of the rank syndrome decoding problem. *IEEE Transactions on Information Theory*, 62(2):1006–1019.

Gaborit, P. and Zémor, G. (2016). On the hardness of the decoding and the minimum distance problems for rank codes. *IEEE Transactions on Information Theory*, 62(12):7245–7252.

Gibson, K. (1995). Severely denting the gabidulin version of the mceliece public key cryptosystem. *Des. Codes Cryptogr.*, 6(1):37–45.

Gibson, K. (1996). The security of the gabidulin public key cryptosystem. In *Proceedings of the 15th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'96, page 212–223, Berlin, Heidelberg. Springer-Verlag.

Hofheinz, D., Hövelmanns, K., and Kiltz, E. (2017). A modular analysis of the fujisaki-okamoto transformation. In Kalai, Y. and Reyzin, L., editors, *Theory of Cryptography*, pages 341–371, Cham. Springer International Publishing.

Loidreau, P. (2017). A new rank metric codes based encryption scheme. In Lange, T. and Takagi, T., editors, *Post-Quantum Cryptography*, pages 3–17, Cham. Springer International Publishing.

McEliece, R. J. (1978). A Public-Key Cryptosystem Based On Algebraic Coding Theory. *Deep Space Network Progress Report*, 44:114–116.

Otmani, A., Kalachi, H. T., and Ndjeya, S. (2018). Improved cryptanalysis of rank metric schemes based on gabidulin codes. *Des. Codes Cryptogr.*, 86(9):1983–1996.

Ourivski, A. V. and Johansson, T. (2002). New technique for decoding codes in the rank metric and its cryptography applications. volume 38, pages 237–246, Cham. Springer International Publishing.

Overbeck, R. (2005). A new structural attack for GPT and variants. In Dawson, E. and Vaudenay, S., editors, *Progress in Cryptology - Mycrypt 2005, First International Conference on Cryptology in Malaysia, Kuala Lumpur, Malaysia, September 28-30, 2005, Proceedings*, volume 3715 of *Lecture Notes in Computer Science*, pages 50–63. Springer.

Overbeck, R. (2008). Structural attacks for public key cryptosystems based on gabidulin codes. *Journal of cryptology*, 21(2):280–301.

Rashwan, H., Gabidulin, E. M., and Honary, B. (2011). Security of the gpt cryptosystem and its applications to cryptography. *Security and Communication Networks*, 4(8):937–946.

Roth, R. (1991). Maximum-rank array codes and their application to crisscross error correction. *IEEE Transactions on Information Theory*, 37(2):328–336.

Saito, T., Xagawa, K., and Yamakawa, T. (2018). Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In Nielsen, J. B. and Rijmen, V., editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 520–551, Cham. Springer International Publishing.

Shehhi, H. A., Bellini, E., Borba, F., Caullery, F., Manzano, M., and Mateu, V. (2019). An ind-cca-secure code-based encryption scheme using rank metric. In Buchmann, J., Nitaj, A., and Rachidi, T., editors, *Progress in Cryptology – AFRICACRYPT 2019*, pages 79–96, Cham. Springer International Publishing.

Shor, P. W. (1997). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509.