Design and Cryptanalysis of Post-Quantum Cryptosystems

Olive Chakraborty

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

THÈSE DE DOCTORANT DE
SORBONNE UNIVERSITÉ

Spécialité

Informatique

École Doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par
OLIVE CHAKRABORTY

Pur obtenir le grade de
DOCTEUR DE SORBONNE UNIVERSITÈ

DESIGN AND CRYPTANALYSIS OF POST QUANTUM CRYPTOSYSTEMS

Thèse dirigée par JEAN-CHARLES FAUGÈRE
et LUDOVIC PERRET

après avis des **rapporteurs**:

Mme. Delaram KAHROBAEI	Professeur, University of York, U.K
M. Jacques PATARIN	Professeur, Université de Versailles

devant le **jury** composé de :

M. Jean-Charles FAUGÈRE	Directeur de recherche, INRIA Paris
M. Stef GRAILLAT	Professeur, Sorbonne Université, LIP6
Mme. Delaram KAHROBAEI	Professeur, University of York, U.K
M. Jacques PATARIN	Professeur, Université de Versailles
M. Ludovic PERRET	Maître de Conférences, Sorbonne Université, LIP6
M. Mohab SAFEY EL DIN	Professeur, Sorbonne Université, LIP6

Résumé

La résolution de systèmes polynomiaux est l'un des problèmes les plus anciens et des plus importants en Calcul Formel et a de nombreuses applications. C'est un problème intrinsèquement difficile avec une complexité, en générale, au moins exponentielle en le nombre de variables. Dans cette thèse, nous nous concentrons sur des schémas cryptographiques basés sur la difficulté de ce problème. Cependant, les systèmes polynomiaux provenant d'applications telles que la cryptographie multivariée, ont souvent une structure additionnelle cachée. En particulier, nous donnons la première cryptanalyse connue du crypto-système « Extension Field Cancellation ». Nous travaillons sur le schéma à partir de deux aspects, d'abord nous montrons que les paramètres de challenge ne satisfont pas les 80 bits de sécurité revendiqués en utilisant les techniques de base Gröbner pour résoudre le système algébrique sous-jacent. Deuxièmement, en utilisant la structure des clés publiques, nous développons une nouvelle technique pour montrer que même en modifiant les paramètres du schéma, le schéma reste vulnérable aux attaques permettant de retrouver le secret. Nous montrons que la variante avec erreurs du problème de résolution d'un système d'équations est encore difficile à résoudre. Enfin, en utilisant ce nouveau problème pour concevoir un nouveau schéma multivarié d'échange de clés nous présentons un candidat qui a été soumis à la compétition Post-Quantique du NIST.
Mots clés : cryptographie, post-quantique, Multivariée, cryptage à clé publique, base de Gröbner, cryptanalyse algébrique, système polynomial avec erreurs, NIST.

Abstract

Polynomial system solving is one of the oldest and most important problems in computational mathematics and has many applications in computer science. It is intrinsically a hard problem with complexity at least single exponential in the number of variables. In this thesis, we focus on cryptographic schemes based on the hardness of this problem. In particular, we give the first known cryptanalysis of the Extension Field Cancellation cryptosystem. We work on the scheme from two aspects, first we show that the challenge parameters don't satisfy the 80 bits of security claimed by using Gröbner basis techniques to solve the underlying algebraic system. Secondly, using the structure of the public keys, we develop a new technique to show that even altering the parameters of the scheme still keeps the scheme vulnerable to attacks for recovering the hidden secret. We show that noisy variant of the problem of solving a system of equations is still hard to solve. Finally, using this new problem to design a new multivariate key-exchange scheme as a candidate for NIST Post Quantum Cryptographic Standards.
Keywords: Post-quantum, Cryptography, Multivariate, Public-key Encryption, Gröbner basis, Algebraic Cryptanalysis, Polynomial systems with Errors, NIST.

To my dearest mother Moushumi and heavenly father Haridash

Acknowledgements

My thesis has only been possible because of a lot of effort, help and support of the people that I came across during this process.

First and foremost, I thank my mother and my heavenly father, it is because of them I am where I am. Without their thankless efforts for all these years nothing of this would have been possible. I am in your debt for my entire life.

I thank my advisors Jean-Charles Faugère and Ludovic Perret for their guidance throughout this journey. I learned an incredible amount of things from them, but in particular how to do research and, more importantly how to deal with roller coaster of emotions that is associated with PhD. They inspired my love for the subjects on which I worked and my decision to pursue an academic career. They are the role models for the scientists that I would like to become.

I would like to thank Jacques Patarin and Deleram Kahrobaei for reviewing this manuscript and for their comments that helped me to improve it. I thank Stef Graillant and Mohan Safey El Din for accepting to be part of the jury of my thesis. Additionally I thank Stef and Jacques again for being a part of my mid PhD evaluation committees and their advice on many topics.

I thank the members of the PolSys, both present and past, for their companionship all these years. In particular, my heartiest thanks to Mohab Safey El Din for his invaluable advice every time I went to him, whether it be academic, administrative or personal. To Jérémy Berthomieu for his unconditional help with every possible thing I can think of (especially teaching me French). I thank my fellow PhD mates, Huu-Phuoc, Xuan, Jocelyn, Eliane, Solane, Nagarjun, Andrew, Jorge and Hieu, for their time shared. I thank the secretaries of our team, lab and école doctoral, for their help all these years.

I would like to thank the CROUS and its staffs who took care of our health providing delicious and healthy food, which I consider is one of the crucial things that allowed me to carry on with my work without worrying about food.

I thank the people that this work gave me who now I proudly call as friends. To Matias, Rachel, James, and Kaie for making my time at work and after it memorable. To Elias Tsigaridas, who I can't thank enough for everything he has done for me during this time and treated me like his own. To Mme. Corado, Rahma, Maurice, Alice, Andrina, Rafa, George, Steph for being the best flatmates ever and making this quarantine a little fun for me.

I thank Saptaparni for being a constant by my side, for her love and support during all this time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The word *cryptography* derives its roots from the two Greek words "$\kappa\rho\upsilon\pi\tau\acute{o}\varsigma$" (hidden) and "$\gamma\rho\alpha\varphi\acute{\eta}$" (writing). Cryptography combines both the science of designing cryptosystems and the science of analyzing the security of the cryptosystems with an effort to break them. Historically, the use of cryptography was limited to ensuring the secrecy of communication. This means guaranteeing two users can communicate over an insecure channel such that no third party can either understand or modify the message. The principal idea of designing a cryptosystem is to modify the message, also called the *plaintext*, such that no one other than the intended receiver can recover the plaintext message from the modified message, which is also known as *ciphertext*.

Over time, cryptography has become the most integral component in information security with such a wide range of applications: secrecy of data, ensuring anonymity, ensuring authenticity of communications and data, etc. Currently, some of the most prominent examples where the use of cryptography is fundamental are web-encryption (HTTPS), end-to-end message encryption (OpenPGP and Whatsapp-Signal protocol), e-money (Bitcoins, Ethereum etc.), ATM and Sim cards (RFIDs) and secure digital-key storage (Hardware Security Module), to name a few.

Cryptography in general can be broadly classified in two types: *symmetric* (or secret-key) and *asymmetric* (or public-key) cryptography. Consider a case when Alice wants to send some message to Bob over some insecure channel. In symmetric cryptography, Alice and Bob initially agree on a shared secret-key. This key is used in both encryption and decryption processes. Some of the famous examples of symmetric cryptosystems include One Time Pads [Sha49], AES [DR99]. One limitation of such cryptosystems is the prior establishment of a secure secret-key that allows for a secure channel of communication. This is answered by public-key cryptography. The idea of such an asymmetric protocol is to securely share the secret-key to the intended recipient such that no third party can get hold of the secret-key even when the information is shared over an insecure channel. The first example of such a scheme can be credited to Diffie and Hellman, who proposed

the Diffie-Hellman Key-exchange protocol [DH76a].

In the asymmetric case, Bob generates two sets of keys, a public-key and a secret-key. For encrypting a message, Alice uses Bob's public-key and sends the encrypted message, i.e. the *ciphertext*, over some channel to Bob. Finally, Bob uses his secret-key to decrypt the ciphertext and recovers the hidden message. Only Bob can recover the message since only Bob has the correct secret-key corresponding to the public-key which generates the ciphertext. This has been depicted in Figure 1.1. Public-key cryptosystem includes a function, which is easy to compute in one way, however, it is hard to invert, unless provided with an additional information, known commonly as *trapdoor*. Such functions are therefore known as *trapdoor one-way functions*. This idea was first introduced by Merkle, Diffie and Hellman in 1976 [DH76a, Mer78]. This concept was also proposed independently by Ellis [Ell70] at `GCHQ`, under the name "non-secret encryption", however, it was not made public until much later. One of the earliest and most important example of public-key cryptosystem is `RSA` [RSA78] which was invented by Rivest, Shamir and Adleman in 1978 (it is essentially the same scheme was also designed by Cocks [Coc73] at `GCHQ` in 1973). Currently, there are many standardized public-key schemes which are available. Majority of the such schemes in practice depend on only three problems:

1. The Integer Factorization Problem (`IFP`) [RSA78, Mon94] : Given $n = pq$, where p and q are primes, find p and q.

2. The Discrete Logarithm Problem (`DLP`) [McC90] : Given α, m and $\beta = \alpha^a \mod n$, find a.

3. The Elliptic Curve Discrete Logarithm Problem (`ECDLP`) [SS98] : Given an elliptic curve E over the finite field $\mathbb{F}_q$ where $q = p^n$ and two points $P, Q \in E(\mathbb{F}_q)$ such that both have the same order, the problem is to find the integer a such that $Q = aP$.

Currently, on a classical computer, there are some algorithms which are known to solve these problems. The General number field sieve algorithm factorizes an integer in time that is sub-exponential in the size of the integer [Car96]. There exists another algorithm that takes quasi-polynomial time to solve the discrete logarithm problem over finite fields of small characteristics [BGJT14]. The best known algorithm which solve the `ECDLP` are based on the parallelized versions of Pollard's Rho algorithm [Pol78, VOW99, Pol00]. The expected running time is dependent on the order of the group, more specifically, $\mathcal{O}(\sqrt{r})$, where r is the order of the points P, Q on the elliptic curve [GG16]. However, the general assumption is that no algorithm exists that can solve all instances of `IFP`,`DLP` and `ECDLP` in polynomial time on a classical (non-quantum) computer.

Classical computers have existed for a long time, however the idea of quantum computing was developed in the 1980's by Paul Benioff, when he proposed

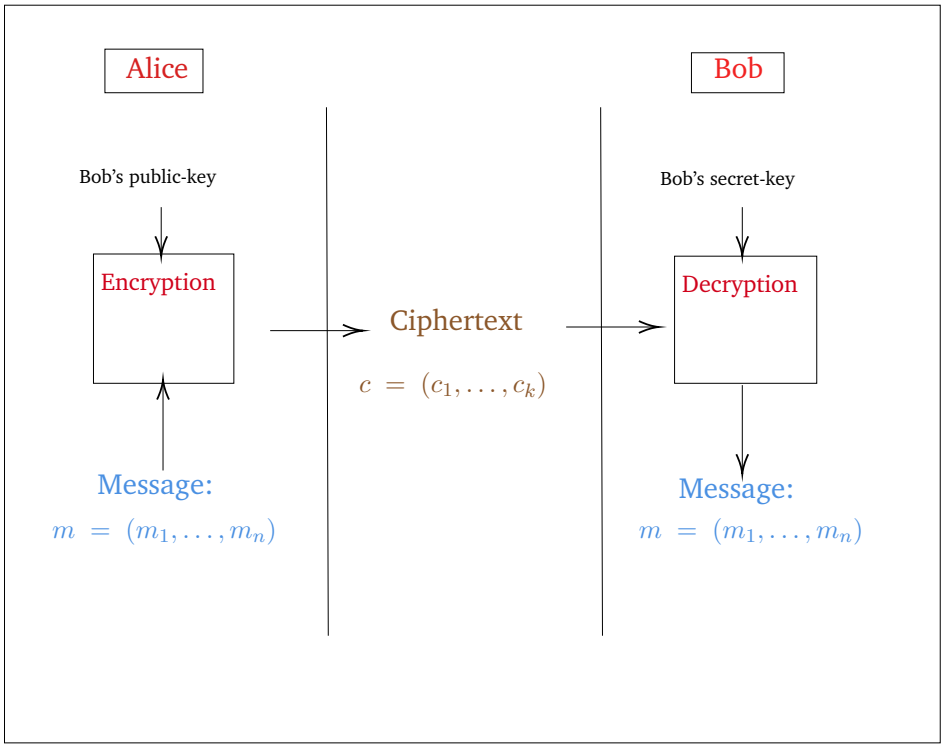

Figure 1.1 – Public-key cryptosystem.

the quantum mechanical model of a Turning machine [Ben80]. It was realised that one could design quantum computers which were able to simulate physical processes that are not possible on a classical computer [Fey82]. Computations on a quantum computer can be to be much faster than on a classical computer. More so, several of the previously mentioned problems can be solved by Shor's algorithm [Sho99] on a quantum computer. Especially, IFP is solved by the Shor's algorithm (with an error probability of maximum 1/3) and requires the use of $\mathcal{O}((\log n)^2(\log\log n)(\log\log\log n))$ number of quantum gates [1], where n is the integer to factorize [Sho99]. Further research by Beauregard has shown that a variant of the Shor's algorithm exists that solves integer factorization using $2n+3$ qubits and $\mathcal{O}(n^3 \log n)$ operations [Bea02]. Recently, [Bea02] showed that Shor's algorithm can solve an instance of ECDLP using $(9n + 2\lceil \log_2 n \rceil + 10)$ qubits and $(448n^3 \log_2 n + 4090n^3)$ quantum gates. Grover's algorithm [Gro96] is another quantum algorithm which essentially provides a speed up on a brute force search from its classical counterpart. Grover's algorithm very briefly can be described as follows: given a function $f : [0, 2^n) \mapsto [0, 2)$, find a $x \in [0, 2^n)$ such that $f(x) = 0$. Grover's algorithm finds a root of f using only about $2^{n/2}$ quantum evaluations of f from a total of 2^n possible inputs. Therefore, this algorithm comes in handy

[1] These gates are the quantum counterparts of the classical logical gates. Refer to Section 2.2.2 for more details.

while mounting brute force attacks against cryptosystems to recover any hidden secret on a quantum computer. The impact of quantum algorithms (such as Shor's and Grover's algorithm) on the security of the current cryptographic standards is detailed in Table 1.1. Even though, so much progress has been made in quantum computing, design and construction of a quantum computer that could replace today's classical computers are not possible as of yet. Nevertheless, in the cryptographic community huge strides are being made in preparation for a time in the future, when such quantum computers become a reality.

Therefore, in 2016 the National Institute of Standards and Technology (`NIST`) announced a call for new post-quantum cryptographic candidates [CCJ+16] with the goal of declaring new standards, to replace the current standardized public-key cryptosystems used in practice e.g., RSA [RSA78], DSA [NIS92] etc. There are mainly five classes of public-key cryptography that are believed to be quantum-resistant: *Multivariate*-based cryptography, *Lattice*-based cryptography, *Code*-based cryptography, *Hash*-based cryptography and *Super-singular Elliptic Curve Isogeny*-based cryptography. In the first round (December 2017) there were 69 submissions of cryptographic primitives. At the time of writing this thesis, 26 submissions have survived through to the second round of the competition.

Name	Function	Pre-quantum sec. level	Post-quantum sec. level
Symmetric cryptography			
AES-128 [DR99]	Encryption	128	64 (*Grover*)
AES-256 [DR99]	Encryption	256	128 (*Grover*)
GMAC [MV04]	MAC	128	128
SHA-256 [Dan15]	Hash	256	128 (*Grover*)
SHA3-256 [BDPA11]	Hash	256	128 (*Grover*)
Asymmetric cryptography			
RSA-3072 [RSA78]	Encryption	128	*Broken*
RSA-3072 [RSA78]	Signature	128	*Broken*
DH-3072 [DH76b]	Key-exchange	128	*Broken*
DSA-3072 [ElG85]	Signature	128	*Broken*
ECDH-256 [Kob87]	Key-exchange	128	*Broken*
ECDSA-3072 [JMV01]	Signature	128	*Broken*

Table 1.1 – Security levels shown are against the best pre-quantum and post-quantum attacks known. Security level b implies that the best attacks use approximately 2^b operations. For hash functions, 'security' in this table refers to pre-image security [BL17].

A very important aspect of narrowing down the field of viable quantum-safe cryptographic primitives involve measuring the hardness for an adversary to break the cryptosystem, or more simply understanding the security of the cryptosystem through *cryptanalysis*. Over time, various cryptanalysis techniques have been developed such as linear cryptanalysis [Mat93], differential cryptanalysis [BS91, FGS05], side-channel cryptanalysis [KSWH98], etc. *Algebraic cryptanalysis* is another method to perform security analysis by reducing the security of the

problem to the hardness of solving a polynomial system of equations. Overall, it can be divided into two steps: The first step involves transforming the cryptosystem's algorithms into a system of multivariate polynomial equations that allows us to recover the secret. After building the system, one estimates the hardness of solving this system. A practical algebraic attack against the cryptosystem if a solution is found to the system of equations.

This problem of solving a multivariate polynomial system of equations, known as the Polynomial System Solving (PoSSo) problem, is NP-Complete [GJ79]. Typically, a random non-linear multivariate system of equations is hard to solve (has exponential complexity). However, in practice, system of equations derived from algebraic modelling of cryptosystems are in general, not random. Algebraic cryptanalysis techniques focus on exploiting the hidden structures of such system of equations, and has resulted in a lot of success over the years [FJ03, CB07, FPS09, BFP08, FJPT10, SK99]. The goal of this thesis is to explore the aspects of algebraic cryptanalysis over multivariate encryption cryptographic primitives and further try designing a new multivariate scheme that are safe from such algebraic attacks.

1.1 Organization and Contributions of the thesis

To present our work, the thesis has been divided into two parts. In the first part, we present the preliminaries for the work of this thesis. In Chapter 2, we present the PoSSo problem on which multivariate cryptography is based. We introduce some state-of-the-art methods to solve this problem. In particular, we focus on algebraic techniques that take use of a mathematical object called Gröbner basis.

In Chapter 3, we give an overview of multivariate cryptography. We introduce the Matsumoto-Imai cryptosystem [MI88], which is one of the first known examples of a multivariate scheme. We also introduce the Hidden Field Equations (HFE) [Pat96]. HFE has provided with the foundation for most of the current multivariate primitives which we also discuss in quite some detail. In particular, we are also interested in one such multivariate encryption scheme, the Extension Field Cancellation cryptosystem [SDP16].

In the second part of the thesis, we present our contributions. More precisely, we address the following topics.

Algebraic cryptanalysis of EFC. The Extension Field Cancellation scheme (EFC) is a recent multivariate public-key cryptosystem that was presented at PQCrypto in 2016. It proposes a new trapdoor for multivariate quadratic cryptographic primitive that allows for encryption, in contrast to most time-tested multivariate trapdoors, like Unbalanced Oil and Vinegar and Hidden Field Equations, which only allow for digital signatures. Numerous multivariate encryption schemes, pro-

posed over the years, have been either broken or have been cryptanalyzed, however, EFC has stood untouched. This motivates us to look at the security of the scheme.

In Chapter 4, we present algebraic attacks against EFC. We report the results of a hybrid Gröbner basis attack [BFP09] on all three challenge parameters of EFC. Using this message recovery attack, for the first and the second challenge parameter we recover the hidden secret message in 2^{65} and 2^{77} operations respectively, which is contrary to the claims of 80 bits of security for these parameters. As previously mentioned, like other multivariate cryptographic schemes, the public-key of EFC also possesses a hidden structure. We provide experimental evidence of the non-random behavior of the public polynomials of EFC. On the EFC scheme with no disregarded public-key polynomials (which we shall see later is called a *minus* version of a multivariate scheme), denoted below as EFC(0), we show that there is a polynomial time attack, polynomial in the number of variables n. This has been stated informally in Theorem 1 below:

Theorem 1 (informal). *Given a public-key* $(f_1, \ldots, f_{2n}) \in \mathbb{F}_q^{2n}[x_1, \ldots, x_n]$ *and the ciphertext* $(c_1, \ldots, c_{2n}) \in \mathbb{F}_q^{2n}$ *from an instance of* EFC(0) *using Gröbner basis, we can recover the hidden secret message in* $\mathcal{O}(n^{3\omega})$ *which is polynomial in* n *and where* $2 \leq \omega < 3$ *is the linear algebra constant.*

We present the full version of Theorem 1 as well as the proof in Section 4.2.4. Extending the idea of this theorem, we explain how a degree 3 linear combination of the public-keys of EFC(0) yield linear equations (see Section 4.2.4 for more details).

We extend this methodology to the minus variant of EFC, denoted as EFC$^-$, where we recover quadratic equations from a high degree (degree ≥ 3) combinations of the public-keys. This technique is quite similar to the approach used against the HFE scheme [FJ03] where the authors show the public-keys exhibiting some algebraic properties are easier to solve than a random system of quadratic equations of the same sizes. We introduce a new technique of explicitly computing and recovering low-degree relations from the public-keys of EFC$^-$. To do so, we consider the initial public-keys and their Frobenius powers. The following Claim 1 informally describes the basic idea.

Claim 1 (informal). *Given the public-keys equations for an instance of* EFC$^-$*, we can always find some combinations of the public-keys and their Frobenius powers which produce new low-degree relations.*

Using this technique, we can recover the quadratic relations from degree 3 combinations in 151 minutes for the first challenge parameter and 110 minutes for the second challenge parameter. This computation is polynomial-time in the number of variables. Furthermore, we show that adding these new equations along with the public equations make the Gröbner basis computation much more efficient as well as reducing the time complexity by a huge factor. For instance, in the case of EFC$^-$ with $n = 75$ and 2 public-key polynomials excluded, adding such intermediate equations reduces the run time of F4 from more than a day to 66.05 seconds to

compute the Gröbner basis. Thus, this scheme has structural weaknesses that can be easily exploited by an adversary to recover secret messages and thus making the scheme unsuitable for encryption.

The PoSSoWN problem. The Leaning With Errors (LWE) problem [Reg09], proposed by Regev in 2009, can be modelled as a problem solving a system of noisy linear equations. Results from [Reg09, BLP+13] have shown that the hardness of this problem can be reduced to the hardness of some of the worst case lattice problems. Naturally, this leads us to the question, whether one can generalize the LWE problem to a non-linear system of noisy equations. In this thesis, we try to answer this exact problem. The non-linear generalization of the LWE problem leads to a noisy variant of the PoSSo problem that we discussed in Chapter 2. We call this problem as the Polynomial System Solving With Noise (PoSSoWN) problem. Particularly, some work in this direction was made in [AFFP11] by introducing the noisy version of the ideal membership problem and the Gröbner basis problem, however, since then not much progress has been made.

In Chapter 5, we describe the PoSSoWN problem. Recalling from Chapter 2 Gröbner bases are mathematical objects that are useful in solving a system of non-linear equations. Interestingly, an algorithm that solves the Gröbner basis problem, which is the problem of computing a Gröbner basis of a system of equations, also solves the PoSSo problem. A variant of the Gröbner basis problem, i.e. the Gröbner basis with Noise problem (GBN), was also introduced and has already been shown to be as hard as the LWE problem [AFFP11]. Naturally, one question arises: can an algorithm that solves the Gröbner basis with Noise (GBN) problem be modelled as an algorithm to solve the PoSSoWN problem ? Or, in other terms, is the PoSSoWN problem NP-Hard?

In this work, we reduce the hardness of PoSSoWN to the LWE problem and the GBN problem for the linear and the non-linear instance of PoSSoWN. To our knowledge, this is the first work which also present the algorithms and the techniques to solve this problem. To solve the problem, one can employ algorithms that solve the PoSSo problem. However, we due to the presence of errors, algorithms presented in Chapter 2 cannot be directly applied. One contribution of this thesis is that we present algorithms to solve any instance of the PoSSoWN problem.

The CFPKM scheme. Since the first multivariate cryptosystem C^* [MI88] was proposed, many schemes based on the PoSSo problem have been designed. In Chapter 5 we presented another hard problem based on solving a polynomial system, called the PoSSoWN. This problem is relatively new in comparison to the PoSSo problem and therefore, hasn't been looked into from the point of view of designing multivariate cryptosystems. The PoSSoWN problem, like the PoSSo problem is another candidate for post-quantum cryptography. This motivated us to design a cryptosystem which relies on the hardness of the PoSSoWN problem. We build

a multivariate public-key cryptosystem for key-exchange, which can be naturally transformed into a public-key Key encapsulation scheme.

In Chapter 6, we present a key-encapsulation scheme, called as `CFPKM`. This scheme was submitted to the `NIST` PQC Standardization competition as a candidate for Public-key Encryption / Key-Encapsulation scheme. We proposed two sets of parameters, `CFPKM128` and `CFPKM182` addressing two security strengths suggested by `NIST`. Unfortunately, this scheme was broken in the second round due to a fault in the design structure of the scheme. Some efforts were made to correct the vulnerability, however, not much progress was made and hence the scheme was dropped.

1.2 Publications

The contribution of Chapter 4 was a joint work with Jean-Charles Faugère and Ludovic Perret. Our results will be appearing in the paper:

- Olive Chakraborty, Jean-Charles Faugère and Ludovic Perret. Cryptanalysis of The Extension Field Cancellation Cryptosystem. In *Design, Codes and Cryptography* (Submitted with minor revisions.)

The contribution of Chapter 6 was also a joint work with Jean-Charles Faugère and Ludovic Perret and was submitted for the NIST Standardization Competition. The whole package of the scheme along with the description and implementation of the scheme is available on the NIST webpage (https://csrc.nist.gov/Projects/post-quantum-cryptography/Round-1-Submissions).

Part I

Preliminaries

Chapter 2

Polynomial System Solving

Abstract

Solving a system of polynomial equations is a fundamental problem in mathematics with a wide range of applications. Cryptography is one such field and is the main focus of this work. Multivariate cryptography relates to cryptosystems which are based on the hardness of solving a system of multivariate polynomial equations over a finite field (the $\texttt{PoSSo}_q$ problem, which is NP-Hard). It is therefore important to understand the cost of solving $\texttt{PoSSo}_q$. In particular, Gröbner basis are mathematical objects that are very useful in solving $\texttt{PoSSo}_q$, which we introduce and describe in great detail. Also in this chapter, we consider combinatorial techniques (such as exhaustive search) for solving $\texttt{PoSSo}_q$.

2.1 General Framework

Throughout this thesis, we use some common notations. Let $\mathbb{F}$ be a field and $\mathbb{F}[x_1, \ldots, x_n]$ be the polynomial ring in n variables $x_1, \ldots, x_n$. In this chapter, we focus our attention to the problem of finding – if any – solution(s) to a system of m algebraic equations in n unknowns over $\mathbb{F}$:

$$\left\{ \begin{array}{c} f_1(x_1, \ldots, x_n) = 0 \\ \vdots \\ f_m(x_1, \ldots, x_n) = 0 \end{array} \right\}$$

This problem of finding the roots of a system of multivariate polynomials is most commonly known as the PoSSo problem. In this work, we deal with system of equations which are defined over finite field of order $q \in \mathbb{N}$ elements (denoted by $\mathbb{F}_q$) where q is some positive power of a prime number. Consequently, we denote this problem as $\texttt{PoSSo}_q$.

$\mathtt{PoSSo}_q$
Input: $f_1, \ldots, f_m \in \mathbb{F}_q[x_1, \ldots, x_n]$
Goal: Find - if any - a vector $(z_1, \ldots, z_n) \in \mathbb{F}_q^n$ such that

$$\left\{ \begin{array}{c} f_1(z_1, \ldots, z_n) = 0 \\ \vdots \\ f_m(z_1, \ldots, z_n) = 0 \end{array} \right\}$$

For linear systems, i.e. degree of each f_i is 1, the problem can be solved in polynomial time, thanks to Gaussian elimination. However, for non-linear cases, the problem is significantly harder to solve and is an NP-Complete [FY79]. When the system of equations is quadratic, this problem is also known as the $\mathtt{MQ}_q$ problem, and is still NP-Complete [FY79]. In the following sections we shall present some techniques to solve the $\mathtt{PoSSo}_q$ in general and the $\mathtt{MQ}_q$ problem in particular.

2.2 Combinatorial Methods

2.2.1 Classical Setting

Since we work with polynomials over a finite field, exhaustive search or brute force search is the most obvious and natural choice for solving a system of polynomials $f_1, \ldots, f_m \in \mathbb{F}_q[x_1, \ldots, x_n]$. This type of combinatorial technique exhaustively searches for values of the variables $(x_1, \ldots, x_n) \in \mathbb{F}_q^n$ such that they satisfy each of the polynomial equations. The complexity of such an algorithm is exponential in the number of variables. Particularly, [BCC+10] details the complexity of a brute force algorithm which computes the solution to a system of quadratic equations in $\mathbb{F}_2$. This has a complexity of $2^{n+2} \cdot \log_2 n$ bit operations.

Example 2.2.1. *We want to compute the common roots of a system of 90 generic quadratic equations over 80 variables in $\mathbb{F}_2[x_1, \ldots, x_{80}]$. Using the exhaustive search method of [BCC+10], the total complexity is 2^{85} binary operations.*

Remark 2.2.2 (A brief remark about the time complexity analysis)**.** *Complexity of algorithms are generally given in terms of Big-Oh notation ($\mathcal{O}()$). For a given positive function $g(x)$, one can denote $\mathcal{O}(g(x))$ the set of functions [CLRS09]*

$$\mathcal{O}(g(x)) = \{f(x) : \textit{there exists positive constants } \mathsf{M} \textit{ and } x_0 \textit{ such that } 0 \leq f(x) \leq \mathsf{M}g(x) \textit{ for all } x \geq x_0\}.$$

This constant M *depends mainly on the algorithm itself. The implementation of algorithm as well as the architecture of the machine on which the algorithm has been implemented also impacts the constant. Some algorithms have large overheads in*

their actual run time on a particular machine, which in turn are reflected in this constant. Therefore in general, the constant M *is not easy to determine. However, estimating the running time without the constants gives an overview of how algorithms compare to each other asymptotically. Thus, even though the constant is not completely precise, as a conservative choice we assume the constant* M *to be 1. In the rest of the work, we analyze the running time of the algorithms with this assumption. In the following theorem (see Theorem 2.2.3), authors of [LPT+17] denote this by* $\mathcal{O}^*$*, which omits the polynomial factors of the* $\mathcal{O}$ *notation.*

Recently, Lokshtanov *et al.* in [LPT+17] proposed a new algorithm for solving a system of non-linear equations which is faster than standard exhaustive search, i.e. $\mathcal{O}(q^n)$. In particular,

Theorem 2.2.3. *Let* p *be a prime, and* $q = p^k$ *for* $k \geq 1$*. There is a randomized algorithm that, given an instance of* m *polynomial equations of degree at most* d *in* n *variables, computes the zeros of the system correctly with high probability. The complexity of the algorithm is*

- $\mathcal{O}^*(2^{0.8765n})$ *when* $q = d = 2$*,*
- $\mathcal{O}^*\left(q^{\left(1-\frac{1}{5d}\right)n} \cdot n^{3d}\right)$ *when* $p = 2$*, but* $q > 2$ *or* $d > 2$*,*
- $\mathcal{O}^*\left(q^{\left(1-(1/200d)\right)n} \cdot n^{3dq}\right)$ *when* $p > 2$ *and* $\log p < 4ed$*,*
- $\mathcal{O}^*\left(q^n \cdot \left(\frac{\log q}{ekd}\right)^{-kn}\right)$ *when* $p > 2$ *and* $\log p \geq 4ed$*,*

where $e = 2.718$ *is the Napier's constant.*

Given a system of polynomial equations $(f_1, \ldots, f_m) \in \mathbb{F}_q^m[x_1, \ldots, x_n]$, the idea is to replace the above system by a single high degree polynomial Q over $n'(< n)$ variables, such that the polynomial Q vanishes on the same zeros as that of the system $f_1, \ldots, f_m$. The intuition is that one can perform an exhaustive search over a smaller number of variables n' instead of n and checks the satisfiability of Q. This gives the algorithm an advantage of not having to guess on a large fraction of possible values. Briefly, the algorithm can be described as follows:

1. Select an integer $n' = \lfloor \delta \cdot n \rfloor$ where $0 < \delta < 1$ depending on d and q.
2. Define a function $P : \mathbb{F}_q^n \mapsto \mathbb{F}_q$ such that $P(\mathbf{x}) = 1 - \prod_{i=1}^{m}(1 - p_i(\mathbf{x})^{q-1})$ where $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{F}_q^n$. Thus for all $\mathbf{a} \in \mathbb{F}_q^n$, $P(\mathbf{a}) = 0$ if and only if $p_1(\mathbf{a}) = \cdots = p_m(\mathbf{a}) = 0$ and $P(\mathbf{a}) = 1$ otherwise.

3. Define a function $Q : \mathbb{F}_q^{n-n'} \mapsto \mathbb{F}_q$ such that

$$Q(x_1, \ldots, x_{n-n'}) = \prod_{\mathbf{a} \in \mathbb{F}_q^{n'}} P(x_1, \ldots, x_{n-n'}, \mathbf{a}),$$

4. We note that there is some $\mathbf{c} \in \mathbb{F}_q^n$ satisfying $P(\mathbf{c}) = 0$ if and only if there is some $\mathbf{b} \in \mathbb{F}_q^{n-n'}$ such that $Q(\mathbf{b}) = 0$.

Performing an exhaustive search on Q allows to recover the same zeros of the initial system with high probability. For more details, we direct the reader to the original paper [LPT+17].

Example 2.2.4. *Following Remark 2.2.2 we approximate $\mathcal{O}^*(2^{0.8765n})$ to be $2^{0.8765n}$. Taking the same example as Example 2.2.1, using this algorithm, the complexity for determining the roots is 2^{71} bit operations.*

2.2.2 Quantum Setting

Similar to the classical case, in the quantum setting the first obvious way to solve a system of polynomial equations is exhaustive search. Thanks to Grover's algorithm [Gro96], a quantum search algorithm, one can achieve a square-root speed up over the classical brute force.

Before we proceed with describing the brute force approach on a quantum computer we need to look at the working of quantum computer very briefly. In the quantum setting, the computations are based on behaviour of subatomic particles. Unlike the classical setting, where information can be represented by two logical states (or bits): either 1 or 0, quantum information is naturally represented by electronic states of an atom [Dir39]. The two main states are the ground state, $|0\rangle$ and excited state, $|1\rangle$. However, as an atom follows laws of quantum mechanics, the general electronic state of an atom is a superposition of the two basic states

$$|\Psi\rangle = a|0\rangle + b|1\rangle,$$

called the quantum bit or *qubit* [Sch95]. Thus $\{|0\rangle, |1\rangle\}$ spans the two dimensional linear vector space for qubit. Similar to logical gates in the classical setting, the quantum analog in quantum computing are the *quantum gates*, eg. Pauli-X gate (which is the quantum equivalent of classical NOT gate but over the inputs $|0\rangle, |1\rangle$). Such gates are reversible[1], unlike some of their their classical counterparts. There are various types of quantum gates that exists which take either one or more qubits as input. Any quantum algorithm is usually represented by a

[1] A logic gate G is reversible if for every possible output y there exists a unique input x such that $G(x) = y$. The input x is a sequence of bits/qubits, whose length is equal to the number of inputs of the gate G.

sequence of quantum gates and is known as a *quantum circuit*.

In [SW16], the authors proposed a quantum algorithm for solving $\texttt{MQ}_2$ problem. The main principle is to perform a fast exhaustive search by using Grover's algorithm. One can solve $(m-1)$ binary quadratic equations in $(n-1)$ binary variables with the Grover's algorithm using a circuit consisting of $(m+n+2)$ qubits and requiring the evaluation of $2^{n/2}\big(2m(n^2+2n)+1\big)$ quantum gates. They also propose a variant for the quantum circuit which in comparison uses $3+n+\lceil \log_2 m \rceil$ qubits but with twice as many quantum gates required.

Example 2.2.5. *Solving 90 binary quadratic equations over 80 variables by exhaustive search with Grover's algorithm thus has expected cost of* 174 *qubits and requires a use of minimum* 2^{60} *quantum gates. Using the variant the expected cost is* 90 *qubits using* 2^{61} *quantum gates.*

2.3 Gröbner Basis

In this thesis, the mathematical object that we use most frequently is Gröbner basis [Buc65]. We will see how the calculation of such a basis makes it possible to solve the PoSSo problem. In this section, we present the notations and the essentials around Gröbner bases that are going to be used in the second part of this thesis.

2.3.1 Preliminary Definitions and Properties

We start by defining two mathematical objects naturally associated with Gröbner bases: *ideals* and *varieties*.

Definition 2.3.1 (Ideal)**.** *[CLO15] An ideal* $I \subseteq \mathbb{F}_q[x_1, \ldots, x_n]$ *is a set of elements such that*

- $0 \in I$,
- *If* $f, g \in I$, *then* $f + g \in I$,
- *If* $f \in I$ *and* $g \in \mathbb{F}_q[x_1, \ldots, x_n]$, *then* $fg \in I$.

We define the ideal generated by the polynomials $(f_1, \ldots, f_m) \in \mathbb{F}_q^m[x_1, \ldots, x_n]$ *as*

$$\langle f_1, \ldots, f_m \rangle := \left\{ \sum_{i=1}^{m} g_i f_i : (g_1, \ldots, g_m) \in \mathbb{F}_q^m[x_1, \ldots, x_n] \right\}.$$

We define the *affine algebraic variety* of $I \subseteq \mathbb{F}_q[x_1, \ldots, x_n]$, denoted by $\mathcal{V}(I)$, as the set of the common zeros of all the polynomials in I, over the algebraically closed finite field $\overline{\mathbb{F}_q}$:

$$\mathcal{V}(I) = \{(a_1, \ldots, a_n) \in (\overline{\mathbb{F}_q})^n | \forall f \in I : f(a_1, \ldots, a_n) = 0\}.$$

When the variety is finite, i.e. $|\mathcal{V}(I)| < \infty$, then the ideal is called zero-dimensional. In this work, we are interested in the set of solutions which belong to $\mathbb{F}_q \subset \overline{\mathbb{F}_q}$ (not in its algebraic closure $\overline{\mathbb{F}_q}$). The set of solutions to the equation $x^q = x$ is the entirety of the field $\mathbb{F}_q$. Thus by appending $x_1^q - x_1, \ldots, x_n^q - x_n \in \mathbb{F}_q[x_1, \ldots, x_n]$ to the input ideal $I = \langle f_1, \ldots, f_m \rangle$, we have

$$\mathcal{V}(\langle f_1, \ldots, f_m, x_1^q - x_1, \ldots, x_n^q - x_n \rangle) = \mathcal{V}(I) \cap \mathbb{F}_q^n,$$

the variety consisting of solutions to the system which lie only in $\mathbb{F}_q$.

To solve a system of equations $(f_1, \ldots, f_m) \in \mathbb{F}_q^m[x_1, \ldots, x_n]$, we calculate the variety, which is denoted by $\mathcal{V}_{\overline{\mathbb{F}_q}}(f_1, \ldots, f_m)$. Any solution to the system of equations also cancels all the polynomials in the ideal $\langle f_1, \ldots, f_m \rangle$. Therefore, the variety $\mathcal{V}(\langle f_1, \ldots, f_m \rangle)$ does not depend on the exact choice of the polynomials $f_1, \ldots, f_m$, rather, it depends only on the ideal generated by these polynomials. Thus, one can try to find another system of polynomial equations that generates the same ideal $I = \langle f_1, \ldots, f_m \rangle$ and are easier to solve than the system $f_1, \ldots, f_m$. Thanks to Gröbner basis, we are able to do so. Informally, Gröbner basis is the generating basis for an ideal that allows to identify in particular the roots of a system as well as deduce many properties of an ideal. Thus, Gröbner basis computation provides us with tools to solve a system of multivariate system of equations. We now look into Gröbner basis in some detail.

We shall recall that a *monomial* in the polynomial ring $\mathbb{F}_q[x_1, \ldots, x_n]$ is a power-product of variables. We write a monomial $x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ as x^α where $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}^n$. The *degree* of a monomial is $\mathtt{deg}(x^\alpha) = \sum_i \alpha_i$. We say that a monomial x^α divides another monomial x^β if and only if for all $1 \leq i \in n$, we have $\alpha_i \leq \beta_i$. This is also denoted as $x^\alpha | x^\beta$.

In the case of polynomial ideals with one variable, the largest term is considered with respect to the order $x^d > x^{d-1} > \cdots > x^2 > x > 1$. Choosing any other term leads to an infinite division process. While dealing with multiple variables, we consider a particular type of total order relation[2] on the set of monomials of $\mathbb{F}_q[x_1, \ldots, x_n]$, which we define as follows:

Definition 2.3.2 (Monomial Ordering [CLO15]). *A monomial ordering on a polynomial ring $\mathbb{F}_q[x_1, \ldots, x_n]$ is a binary relation $\succ$ on $\mathbb{N}^n$ such that*

[2] A total order is a type of binary relation on a set which has three principal properties: antisymmetry ($a \leq b$ & $b \leq a \implies a = b$), transitivity ($a \leq b$ & $b \leq c \implies a \leq c$) and connexity ($a \leq b$ or $b \leq a$)

- $\succ$ *is a total ordering on* $\mathbb{N}^n$.
- *For a triplets of monomials* $(x^\alpha, x^\beta, x^\gamma)$*;*

$$x^\alpha \succ x^\beta, \text{ implies } x^\alpha x^\gamma \succ x^\beta x^\gamma,$$

where $\alpha, \beta, \gamma \in \mathbb{N}^n$.

- $\succ$ *is a well-ordering [CLO15, Lem. 2.2.2] on* $\mathbb{N}^n$*, that is every non-empty subset of* $\mathbb{N}^n$ *has a minimal element with respect to* $\succ$.

For instance, the *Lexicographic* (LEX) and *Degree Reverse Lexicographic* (GREVLEX) - which are widely used in practice- are defined as follows:

Definition 2.3.3. *Let* $\succ$ *be a monomial ordering such that* $x_1 \succ x_2 \succ \cdots \succ x_n$*. Let* $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}^n$ *and* $\beta = (\beta_1, \ldots, \beta_n) \in \mathbb{N}^n$

- ***Lexicographic ordering*** (LEX)*: we say* $x^\alpha \succ_{\text{LEX}} x^\beta$*, if and only if there is* $1 \leq k \leq n$ *such that*

$$\begin{cases} (\forall\ 1 \leq i < k) & \alpha_i = \beta_i, \\ & \alpha_k > \beta_k. \end{cases}$$

- ***Graded Reverse Lexicographic ordering*** (GREVLEX)*: given two monomials* x^α *and* x^β*, we say* $x^\alpha \succ_{\text{GREVLEX}} x^\beta$*, if and only if,*

$$|\alpha| = \sum_{i=1}^{n} \alpha_i > |\beta| = \sum_{i=1}^{n} \beta_i, \quad or,$$

$$|\alpha| = |\beta| \text{ and } \exists\ k \text{ such that } (\forall\ i > k)\ \alpha_i = \beta_i \text{ and } \alpha_k < \beta_k.$$

Example 2.3.4. *Consider the* LEX *ordering* $\succ_{\text{Lex}}$ *on* $\mathbb{F}_q[x, y]$*, such that* $x \succ_{\text{Lex}} y$*. Then* $x^3 \succ_{\text{Lex}} xy^2$*,* $x \succ_{\text{Lex}} y^{50}$ *and* $xy^3 \succ_{\text{Lex}} xy^2$.

Example 2.3.5. *Consider the* GREVLEX *ordering* $\succ_{\text{GRevLex}}$ *on* $\mathbb{F}_q[x, y, z]$*, such that* $x \succ_{\text{GRevLex}} y$ *and* $y \succ_{\text{GRevLex}} z$*. Then* $xy^2 \succ_{\text{GrevLex}} x^2z$*, and* $y^{50} \succ_{\text{GRevLex}} x$.

There are many other monomial orderings which exist and refer the reader to [CLO15] for more details. Now, that we have defined ordering amongst monomials, it is easy to note that any polynomial $f \in \mathbb{F}_q[x_1, \ldots, x_n]$ has a unique leading term. Hereby, we provide its formal definition:

Definition 2.3.6 (Leading Monomial, Coefficient and Term)**.** *Let* $f = \sum_{\alpha \in \mathbb{N}^n} c_\alpha x^\alpha \in \mathbb{F}_q[x_1, \ldots, x_n]$ *be a non zero polynomial and let* $\succ$ *be the monomial ordering, then*

- *The leading monomial of* f *with respect to* $\succ$*, denoted by* $\texttt{LM}_\succ(f)$*, is the largest monomial (with respect to* $\succ$*), i.e.*

$$\texttt{LM}_\succ(f) := \texttt{max}\{x^\alpha : c_\alpha \neq 0\},$$

- *The leading coefficient of f with respect to $\succ$, denoted by $\mathtt{LC}_\succ(f)$, is the coefficient associated to the leading monomial of f , i.e.*

$$\mathtt{LC}_\succ(f) := c_\alpha \text{ such that } \mathtt{LM}_\succ(f) = x^\alpha,$$

- *The leading term of f, denoted by $\mathtt{LT}_\succ(f)$, is the product of the leading monomial and coefficient of f, i.e.*

$$\mathtt{LT}_\succ(f) := \mathtt{LC}_\succ(f)\mathtt{LM}_\succ(f).$$

Next, we shall define the notion of a particular type of ideal which can be generated by a set of monomials. This is known as *monomial ideal*.

Definition 2.3.7 (Monomial Ideal)**.** *We say that an ideal $I' \subset \mathbb{F}_q[x_1, \ldots, x_n]$ is a monomial ideal if I' can be generated by a family of monomials, i.e. $I' = \langle x^{\alpha^{(1)}}, \cdots, x^{\alpha^{(m)}} \rangle$ where $\alpha^{(i)} \in \mathbb{N}^n$.*

Example 2.3.8. *An example of monomial ideal is given by $I = \langle x^4y^2, x^3y^4, x^2y^5 \rangle \subset \mathbb{F}_q[x, y]$.*

A monomial x^β belongs to the monomial ideal $I' = \langle x^{\alpha^{(1)}}, \cdots, x^{\alpha^{(m)}} \rangle$, if and only if x^β is divisible by $x^{\alpha^{(i)}}$. Additionally, a polynomial f belongs to monomial ideal I', if and only if all monomials that occur in f with non-zero coefficient also belong to I'.

A special kind of monomial ideal is the ideal generated by the leading monomials of the polynomials (see Definition 2.3.6). This ideal is called the *initial ideal* and we formally define it as follows:

Definition 2.3.9 (Initial Ideal)**.** *Let $\succ$ be a monomial ordering and $I \subseteq \mathbb{F}_q[x_1, \ldots, x_n]$ be an ideal. Then the initial ideal of I, denoted by $\mathtt{LM}_\succ(I)$, is the monomial ideal generated by the leading monomials of the all the polynomials in I, i.e.,*

$$\langle \mathtt{LM}_\succ(I) \rangle := \langle \mathtt{LM}_\succ(f) : f \in I \rangle.$$

When I is already a monomial ideal then $\langle \mathtt{LM}_\succ(I) \rangle = I$. Now, from definition, $\langle \mathtt{LM}_\succ(I) \rangle$ is generated by the monomials $\mathtt{LM}_\succ(f)$ for $f \in I - \{0\}$. Dickson's Lemma [CLO15, Theorem 2.4.5] states that a monomial ideal I has a finite basis. Using this property, one can show that for any polynomial ideal, $I \subseteq \mathbb{F}_q[x_1, \ldots, x_n]$, there exists a finite basis $(g_1, \ldots, g_m) \in \mathbb{F}_q^m[x_1, \ldots, x_n]$ of I, which has the property, $\langle \mathtt{LM}_\succ(I) \rangle = \langle \mathtt{LM}(g_1), \ldots, \mathtt{LM}(g_m) \rangle$. This is most famously known as the *Hilbert Basis Theorem* [CLO15, Theorem 2.5.4]. Any basis which satisfies such a property is called the *Gröbner basis* of the ideal I. We define it more formally as follows:

Definition 2.3.10 (Gröbner basis). *Let $\succ$ be a monomial ordering on the polynomial ring $\mathbb{F}_q[x_1, \ldots, x_n]$. A finite subset G of an ideal $I \subseteq \mathbb{F}_q[x_1, \ldots, x_n]$ is a Gröbner basis of I with respect to $\succ$ if and only if*

$$\langle \mathtt{LM}_{\succ}(I) \rangle = \langle \{ \mathtt{LM}_{\succ}(g) : g \in G \} \rangle.$$

Equivalently, we also say that G is a Gröbner basis if, for every $f \in I$, there exist some $g \in G$ such that $\mathtt{LM}(g) \mid \mathtt{LM}(f)$.

From a practical point of view, computing a LEX Gröbner basis much slower than computing a Gröbner basis with respect to another monomial ordering. On the other hand, it is quite well known that computing GREVLEX Gröbner bases is much faster in practice.

One might encounter a case where there exists a polynomial in the Gröbner basis $g \in G$, such that its leading monomial can be generated by the leading monomials of the other elements in the basis G. Then the basis $G - \{g\}$ is also a Gröbner basis for the same ideal I [CLO15, Lemma 2.7.4]. Removing all such dependent $g \in G$ having this property leads us to the notion of a *minimal* Gröbner basis. However, for an ideal I, one can encounter multiple minimal Gröbner bases. Fortunately, we can find a minimal basis, with the additional property that for any element $g \in G$, no monomial of g lie in monomial ideal $\langle \mathtt{LM}(G - \{g\}) \rangle$. This is the notion of *reduced Gröbner basis* that we formally define below.

Definition 2.3.11 (Reduced Gröbner basis [CLO15]). *A Gröbner basis G for some ideal in the polynomial ring $\mathbb{F}_q[x_1, \ldots, x_n]$ is said to be reduced Gröbner basis if and only if*

- *every polynomial in G is monic, i.e. $\forall g \in G$, $\mathtt{LC}_{\succ}(g) = 1$, and*
- *$\forall g \in G$, no monomial appearing in g belongs to $\langle \mathtt{LM}_{\succ}(G - \{g\}) \rangle$*

For any ideal $I \neq 0$, such a reduced Gröbner basis is always unique [CLO15, Proposition 2.7.6].

While working with Gröbner bases, broadly two types of polynomial systems are encountered: *homogeneous* and *affine* system of equations.

Definition 2.3.12. *Given a multivariate polynomial $f \in \mathbb{F}_q[x_1, \ldots, x_n]$, it is said to be **homogeneous** if and only if all the monomials of f with non-zero coefficients have the same total degree, i.e., with $(\alpha_1, \ldots, \alpha_n) \in \mathbb{N}$, all monomials of f are of the form $x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$ such that $\sum_{i=1}^{n} \alpha_i$ is a constant value for the polynomial f. Otherwise its called an **affine polynomial**.*

As we shall see in Section 2.3.2, one of the most common aspect in Gröbner basis algorithms is the idea of incremental degree by degree computation of the

basis. More precisely, such algorithms consider all polynomials at a certain degree in order to find the generating elements for the Gröbner basis at that degree before proceeding to the next degree. This is made possible by considering the ideal subset, $I_{\leq d} \subseteq I \subseteq \mathbb{F}_q[x_1, \ldots, x_n]$, such that $I_{\leq d}$ consists of all the polynomials in the ideal whose degree is less than or equal to d. We also call this as the degree d-truncated ideal. Using this notion of degree truncated ideal, we define the generating Gröbner basis for this ideal $I_{\leq d}$ as follow:

Definition 2.3.13 (d-Gröbner basis)**.** *Let $\succ$ be a monomial ordering, $d \in \mathbb{N}$ be an integer and $I \subseteq \mathbb{F}_q[x_1, \ldots, x_n]$ be an ideal. A degree d truncated Gröbner basis for I with respect to the monomial ordering $\succ$ is a finite set $G_d \subset I_{\leq d} \subseteq I$ such that for every $f \in I_d$ with $\mathtt{deg}(f) \leq d$, we have $\mathtt{LM}_{\succ}(f) \in \langle \{\mathtt{LM}_{\succ}(g) : g \in G_d\} \rangle$*

And as a direct consequence of the Ascending Chain condition [CLO15, Theorem 2.5.7] and Hilbert Basis theorem [CLO15, Theorem 2.5.4], we have the following theorem.

Theorem 2.3.14. *Let $G_d \subset \mathbb{F}_q[x_1, \ldots, x_n]$ be a degree d-Gröbner basis of a system of homogeneous polynomials in the polynomial ring $\mathbb{F}_q[x_1, \ldots, x_n]$ with respect to some monomial ordering $\succ$. Then we have the inclusion of truncated Gröbner bases with incremental degree and we can find a D such that*

$$G_2 \subset G_3 \subset \cdots \subset G_D = G_{D+1} = G,$$

where $G \subset \mathbb{F}_q[x_1, \ldots, x_n]$ is the Gröbner basis of input system of polynomials with respect to $\succ$.

The step of computing the basis is usually the most difficult step as generally the input polynomials have no mathematical structure. This notion of truncated Gröbner basis comes in handy to provide some kind of structure to this [Fau99]. We shall see later (Section 2.3.2), state-of-the-art Gröbner basis algorithms such as Buchberger and F4 incrementally solve a system of equations by computing Gröbner basis degree by degree.

2.3.1.1 Gröbner Basis and Ideal Membership

One of the important applications of Gröbner basis is that it allows to solve the *Ideal Membership Problem*. Formally, given a polynomial and an ideal, the decision Ideal Membership Problem decides whether the polynomial belongs to the ideal. The testing for ideal membership requires an understanding of the notion of the polynomial division with respect to a set of polynomials. One might recall, division of a polynomial by another polynomial iterates the process of division by the divisor polynomial until the leading term of the remainder in each step of division is not divisible by the leading term of the divisor. The following theorem gives the general form of the division algorithm of a polynomial by an ordered set by building on the previous algorithm.

Theorem 2.3.15. *[CLO15, Theorem 2.3.3] Let $F = (f_1, \ldots, f_m) \in \mathbb{F}_q^m[x_1, \ldots, x_n]$ be a m-tuple of polynomials and let $\succ$ be a fixed monomial ordering. Then every $f \in \mathbb{F}_q[x_1, \ldots, x_n]$ can be written as*

$$f = a_1 f_1 + \cdots + a_m f_m + r,$$

where $a_i, r \in \mathbb{F}_q[x_1, \ldots, x_n]$. We call r the remainder of the division of f by F. The remainder is either $r = 0$ or a linear combination, with coefficients in $\mathbb{F}_q$, of monomials none of which are divisible by any of $\mathtt{LM}_\succ(f_1), \ldots, \mathtt{LM}_\succ(f_m)$.

Example 2.3.16. *Let $f = xy^2 + x^2y \in \mathbb{F}_q[x, y]$ where q is a large prime. Dividing f by $f_1 = xy - 1, f_2 = y + 1 \in \mathbb{F}_q[x, y]$, the division algorithm described above gives us*

$$f = (y + x) \cdot f_1 + 1 \cdot f_2 + x$$

Having $r = 0$ is a sufficient condition for testing the ideal membership of a polynomial $f \in \mathbb{F}_q[x_1, \ldots, x_n]$ in an ideal $I \subseteq \mathbb{F}_q[x_1, \ldots, x_n]$. However, it is not a necessary condition. Consider the following example.

Example 2.3.17. *Let $f = xy^2 - x \in \mathbb{F}_q[x, y]$ and let $f_1 = xy + 1, f_2 = y^2 - 1 \in \mathbb{F}_q[x, y]$ be the two divisors. Dividing f by $F = (f_1, f_2)$ in this particular order gives us*

$$xy^2 - x = y \cdot (xy + 1) + 0 \cdot (y^2 - 1) + (-x - y).$$

While with the choice of $F = (f_2, f_1)$ we have

$$xy^2 - x = x \cdot (y^2 - 1) + 0 \cdot (xy + 1) + 0.$$

From this above example we see that $r = 0$ when the choice of divisors is (f_2, f_1) and thus $f \in \langle f_1, f_2 \rangle$. However, the with the choice of order of divisors as (f_1, f_2), we have $r \neq 0$. Therefore, one might look for a better generator set of the ideal such that with just $r = 0$ we have a necessary and sufficient condition for ideal membership testing. Gröbner basis properties allow the remainder of a polynomial division by the ideal to be uniquely determined [CLO15, Proposition 2.6.1], thus making $r = 0$ a necessary as well as a sufficient condition for ideal membership.

Definition 2.3.18 (Normal Form)**.** *[CLO15, Page 82] Let $I \subseteq \mathbb{F}_q[x_1 \ldots, x_n]$ be an ideal and G be a Gröbner basis of I. Then any polynomial $f \in \mathbb{F}_q[x_1, \ldots, x_n]$ can be represented as $f = h + r$ where $h \in I$ and $r \in \mathbb{F}_q[x_1, \ldots, x_n]$ has no monomials that are divisible by any of $\mathtt{LM}_\succ(g_1), \ldots, \mathtt{LM}_\succ(g_m)$ for $G = (g_1, \ldots, g_m) \in \mathbb{F}_q^m[x_1, \ldots, x_n]$. This polynomial r is called the **normal form** of f with respect to I and $\succ$. We denote it as $\overline{f}^G$.*

Now, with this we can represent any polynomial uniquely by their normal form. Since this normal form is unique for any polynomial, we now have a necessary and sufficient condition to test the ideal membership. More formally we have the following:

Corollary 2.3.19 (Test for Ideal Membership). *Let G be a Gröbner basis for an ideal $I \subseteq \mathbb{F}_q[x_1, \ldots, x_n]$. A polynomial $f \in \mathbb{F}_q[x_1, \ldots, x_n]$ belongs to the ideal I, if and only if the normal form of f with respect to the Gröbner basis is 0, i.e. $\overline{f}^G = 0$.*

Now, that we have discussed the testing for ideal membership, we now focus on the problem of deciding whether an input generating set of an ideal is a Gröbner basis. A generating set $(f_1, \ldots, f_m) \in \mathbb{F}_q^m[x_1, \ldots, x_n]$ for an ideal cannot be a Gröbner basis if the leading term of any polynomial combination of the generator polynomials is not in the ideal $\langle \mathtt{LM}_\succ(f_i) \rangle$ for some fixed monomial ordering $\succ$. This can occur in cases when the leading terms in the combination cancel, leaving only smaller terms, which are not divisible by any of $\mathtt{LM}_\succ(f_1), \ldots, \mathtt{LM}_\succ(f_m)$. Such a combination of two polynomials is known as the *S-polynomial* of a pair of polynomials:

Definition 2.3.20 (S-polynomial). *Let $f, g \in \mathbb{F}_q[x_1, \ldots, x_n]$ be two non-zero polynomials. Let us denote the least common multiple of $\mathtt{LM}_\succ(f)$ and $\mathtt{LM}_\succ(g)$ by x^γ. The **S-polynomial** of f and g is the combination*

$$\mathtt{Spol}_\succ(f, g) := \frac{\mathtt{LT}_\succ(g)}{x^\gamma} \cdot f - \frac{\mathtt{LT}_\succ(f)}{x^\gamma} \cdot g.$$

Using S-polynomials, Buchberger proposes the following decision criteria to determine if the basis (the generating set) of an ideal is a Gröbner basis.

Theorem 2.3.21 (Buchberger's Criterion). *[Buc76, Theorem 3.3] Consider a polynomial ideal $I \subseteq \mathbb{F}_q[x_1, \ldots, x_n]$. Then a basis $G = (g_1, \cdots, g_m) \subset I$ is a Gröbner basis of I with respect to monomial ordering $\succ$ if and only if for all pairs $(g_i, g_j) \in \mathbb{F}_q^2[x_1, \ldots, x_n]$ with $i \neq j$, the remainder on division of $\mathtt{Spol}_\succ(g_i, g_j) \in \mathbb{F}_q[x_1, \ldots, x_n]$ by G is zero.*

Thus using this previous criterion one can test whether a given basis is a Gröbner basis. In the following section, we can now describe the state-of-the-art algorithms to compute Gröbner bases for a system of polynomials.

2.3.2 Gröbner Basis Algorithms

2.3.2.1 Buchberger Algorithm

Buchberger [Buc76] proposed the first general algorithm to compute the Gröbner basis by using the criterion of Theorem 2.3.21. Now we present the algorithm in brief. Given a system of polynomials, say $L \subset \mathbb{F}_q[x_1, \ldots, x_n]$, belonging to an ideal, the goal is to decide whether this set, L is a Gröbner basis for the ideal, and if not, then compute the Gröbner basis. The idea of the algorithm is as follows:

1. Find all S-polynomials (see Definition 2.3.20) for every pair of polynomials in the list.

2. For each S-polynomial, compute the remainder on its division by L. If the remainder is non-zero, then append the remainder to L.

3. Use the condition in Theorem 2.3.21 to check whether this intermediate list of polynomials, L is a Gröbner basis. If not, then go to step 1 by recomputing new sets of S-polynomials from the updated list of polynomials.

We present the Buchberger's algorithm more formally in Algorithm 1. The termination of this algorithm is guaranteed by the ascending chain condition of ideals [CLO15, Theorem 2.5.7]. The addition of each new generator strictly increases the size of the ideal generated by the leading monomials of the generators.

Algorithm 1 Buchberger's Algorithm

Input: A list of polynomials $(f_1, \ldots, f_m) \in \mathbb{F}_q^m[x_1, \ldots, x_n]$ and a monomial ordering $\succ$.

Output: Gröbner basis G of $\langle f_1, \ldots, f_k \rangle$ with respect to $\succ$.

$G \leftarrow \{f_1, \ldots, f_k\}$
$G' \leftarrow \{(f_i, f_j) \in G \text{ such that } f_i \neq f_j\}$;
while $G' \neq \emptyset$ **do**
    $(f, g) \leftarrow \text{Select}(G')$
    $G' \leftarrow G' \backslash \{(f, g)\}$
    $r \leftarrow$ Remainder of the division of $\texttt{Spol}_{\succ}(f, g)$ with respect to G.
    if $r \neq 0$ **then**
        $G \leftarrow G \cup \{r\}$
    end if
    $G' \leftarrow G' \cup \{(g_i, r) : \forall g_i \in G \backslash \{r\}\}$
end while
return G

2.3.2.2 Macaulay Matrices

Macaulay matrices are mathematical objects that are useful in representing the bases of an ideal represented as a vector space. In particular, the basis elements are represented in a matrix form, which allows to takes use of linear algebra methods for manipulating the elements of the basis and obtain a Gröbner basis. Before we proceed, we shall formally define a Macaulay matrix. To define Macaulay matrices [Mac02], we shall use the following notations. Let $\mathbb{F}_q[x_1, \ldots, x_n]$ be a polynomial ring and $\succ$ be a monomial order. We denote $M(d)$ to be the set of all the monomials of degree less than or equal to d and $\mu_i^{\leq d}$ be the i^{th} element of $M(d)$ ordered with respect to the ordering $\succ$. We denote $\ell_d = \binom{n+d}{d}$ as the total number

of monomials in $M(d)$. Finally, for $f \in \mathbb{F}_q[x_1, \ldots, x_n]$, we denote $\text{Coeff}(f, \mu_i^{\leq d})$ the coefficient of f associated with the monomial $\mu_i^{\leq d}$.

Definition 2.3.22 (Macaulay Matrix)**.** *Let $\succ$ be a monomial ordering. Let $F = (f_1, \ldots, f_m) \in \mathbb{F}_q^m[x_1, \ldots, x_n]$ be a list of polynomials of degrees $d_1, \ldots, d_m$ respectively. The Macaulay matrix of F in degree $d \in \mathbb{N}$ is the matrix $\mathcal{M}_{\succ,d}(F) \in \mathbb{F}_q^{\left(\sum_i^m \binom{n+d-d_i}{d-d_i}\right) \times \binom{n+d}{d}}$ where*

- *Each of the $\binom{n+d}{d}$ columns of $\mathcal{M}_{\succ,d}$ is indexed by a monomial of $\mathbb{F}_q[x_1, \ldots, x_n]_d$ of degree less than d. The columns are sorted in decreasing order with respect to the monomial ordering $\succ$.*
- *We denote by $\mu^{\leq d}$ as the ordered set of monomials of degree less than or equal to d, occurring in $\mathbb{F}_q[x_1, \ldots, x_n]$ and $\mu_j^{\leq d}$ as the j^{th} element of that set. Each row of the Macaulay matrix $\mathcal{M}_{\succ,d}(F)$ is indexed by a pair $(f_i, \mu_k^{\leq d-d_i})$, where $i \in \{1, \ldots, m\}$ and $\mu_k^{\leq d-d_i} \in \mathbb{F}_q[x_1, \ldots, x_n]$ where $1 \leq k \leq \ell_{d-d_i}$ where ℓ_{d-d_i} is the number of monomials of degree $d - d_i$.*
- *The element in the row indexed by $(f_i, \mu_k^{\leq d-d_i})$ and the column indexed by $\mu_j^{\leq d}$ corresponds to the coefficient of the monomial $\mu_j^{\leq d}$ in the polynomial $\mu_k^{\leq d-d_i} f_i$.*

$$
\begin{array}{c}
 \\
\mu_1^{\leq (d-d_1)} f_1 \\
\mu_2^{\leq (d-d_1)} f_1 \\
\vdots \\
\mu_{\ell_{d-d_1}}^{\leq (d-d_1)} f_1 \\
\vdots \\
\mu_1^{\leq (d-d_m)} f_m \\
\vdots \\
\mu_{\ell_{d-d_m}}^{\leq (d-d_m)} f_m
\end{array}
\begin{array}{c}
\begin{array}{ccccccc}
 & \mu_1^{\leq d} \succ & & \mu_2^{\leq d} \succ & & \cdots & \mu_{l_d}^{\leq d}
\end{array} \\
\begin{bmatrix}
\cdots & \cdots & & \cdots & \cdots & \cdots & \\
\cdots & \cdots & & \cdots & \cdots & \cdots & \\
 & & & & & & \\
\cdots & \cdots & & \cdots & \cdots & \cdots & \\
 & \cdots & \text{Coeff}\left(\mu_k^{\leq (d-d_j)} f_j, \mu_i^{\leq d}\right) & & & & \\
\cdots & \cdots & & \cdots & \cdots & \cdots & \\
\cdots & \cdots & & \cdots & \cdots & \cdots & \\
\cdots & \cdots & & \cdots & \cdots & \cdots &
\end{bmatrix}
\end{array}
$$

Let $F = (f_1, \ldots, f_m) \in \mathbb{F}_q^m[x_1, \ldots, x_n]$. We can see that linear combination of the rows of $\mathcal{M}_{\succ,d}(F)$ represents a polynomial $f \in \mathbb{F}_q[x_1, \ldots, x_n]$ of degree at most d in the ideal $I_{\leq d}$ generated by F, i.e. $f \in \langle F \rangle_d$. Also row and column operations on $\mathcal{M}_{\succ,d}(F)$ represents elements from the degree d truncated ideal $\langle F \rangle$. For instance, multiplying a non zero constant c by the row with indexed by (f_i, μ_j), we obtain a row which corresponds to the coefficients of $c\mu_j f_i \in \mathbb{F}_q[x_1, \ldots, x_n]$. Similarly adding two rows indexed by (f_i, μ_j) and (f_k, μ_l), we obtain a row that corresponds to the coefficients of the polynomial $\mu_j f_i + \mu_l f_k \in \mathbb{F}_q[x_1, \ldots, x_n]$. Thus we can

say that the matrix $\mathcal{M}'_{\succ,d}(F)$ resulting from some linear algebra operations on the rows of $\mathcal{M}_{\succ,d}(F)$ represents polynomials in $\langle F\rangle_d$.

A connection between the degree d Macaulay matrices $\mathcal{M}_{\succ,d}$ and truncated d-Gröbner basis for an ideal $I_{\leq d}$ was first provided by Lazard in [Laz83]. For a matrix $M \in \mathbb{F}_q^{m\times n}$, we denote by $\widetilde{M}$ the Gauss-Jordan elimination of M. It is also known as the **row echelon form** of the matrix M.

Lemma 2.3.23 ([Laz83]). *Given a system of homogeneous polynomials $F = (f_1, \ldots, f_m) \in \mathbb{F}_q[x_1, \ldots, x_n]$, then $\widetilde{\mathcal{M}_{\succ,d}(F)}$ represents a degree d truncated (non-reducible) Gröbner basis of F.*
Additionally, for any system of polynomials $F' = (f'_1, \ldots, f'_m) \in \mathbb{F}_q[x_1, \ldots, x_n]$, there exists a $d \in \mathbb{N}$, such that the rows of $\widetilde{\mathcal{M}_{\succ,d}(F')}$ form a Gröbner basis for F'.

Using this lemma, Lazard's algorithm successively computes the truncated non-reducible Gröbner basis from $\widetilde{\mathcal{M}_{\succ,1}(F)}, \ldots, \widetilde{\mathcal{M}_{\succ,D}(F)}$, where D is the degree at which the truncated basis is the Gröbner basis for F.

Algorithm 2 Lazard's Algorithm

Input: A list of homogeneous polynomials $F := (f_1, \ldots, f_m) \in \mathbb{F}_q^m[x_1, \ldots, x_n]$ and a monomial ordering $\succ$ and a degree D.
Output: Truncated degree D-Gröbner basis G_D of $\langle f_1, \ldots, f_k\rangle$ with respect to $\succ$.

1: $G_0 \leftarrow \{\}$
2: **for** d **from** 1 **to** D **do**
3: $\quad \mathcal{M}_{\succ,d}(F) \leftarrow$ Macaulay matrix for F w.r.t $\succ$ at degree d
4: $\quad \widetilde{\mathcal{M}_{\succ,d}(F)} \leftarrow$ Reduced row echelon form of $\mathcal{M}_{\succ,d}(F)$
5: $\quad P_d \leftarrow$ Non-zero polynomials represented by the rows of $\widetilde{\mathcal{M}_{\succ,d}(F)}$
6: $\quad G_d \leftarrow G_{d-1} \cup \{g \in P_d : (\forall h \in G_{d-1})\mathtt{LM}_{\succ}(h)$ does not divide $\mathtt{LM}_{\succ}(g)\}$
7: **end for**
8: **return** G_D

2.3.2.3 Faugère's F4 Algorithm

The Buchberger's algorithm involves making the following choices :

1. the choice of a pair of polynomials from the list of pairs of polynomials in the input.

2. the choice of a divisor from a list of divisors while dividing the $\mathtt{Spol}_{\succ}$ by a list of polynomials.

The choices made during the process of Buchberger's algorithm although does not impact the correctness of the algorithm, but greatly dominates the running time of

the Gröbner basis computation [Buc65]. Thus the problem of making a decision on the selection strategy becomes important. Jean-Charles Faugère proposed a new algorithm, called the `F4` [Fau99], which rather than considering just one pair, the algorithm takes into account a set of polynomials pairs at the same time. These pair of polynomials are known as *critical pairs*, which we formally define as follows:

Definition 2.3.24 (Critical Pairs)**.** *A critical pair of two polynomials* $(f_i, f_j) \in \mathbb{F}_q^2[x_1, \ldots, x_n]$ *is an element*

$$\texttt{Pair}(f_i, f_j) := (lcm_{i,j}, t_i, f_i, t_j, f_j) \in \texttt{T}^2 \times \mathbb{F}_q[x_1, \ldots, x_n] \times \texttt{T} \times \mathbb{F}_q[x_1, \ldots, x_n],$$

where

$$lcm_{i,j} = lcm(f_i, f_j) = \texttt{LT}(t_i f_i) = \texttt{LT}(t_j f_j) = lcm(\texttt{LT}(f_i), \texttt{LT}(f_j)),$$

and `T` *is the set of all terms over* $\mathbb{F}_q[x_1, \ldots, x_n]$.

The degree of a critical pair $p_{i,j} = \texttt{Pair}(f_i, f_j)$ is $\deg(p_{i,j})$ which is equal to $\deg(\text{lcm}_{i,j})$. Finally we use two other functions which are defined as $\texttt{Left}(p_{i,j}) := (t_i, f_i)$ and $\texttt{Right}(p_{i,j}) := (t_j, f_j)$, where $t_i, t_j \in \texttt{T}$, the set of all terms over $\mathbb{F}_q[x_1, \ldots, x_n]$. With these we can now describe `F4` in Algorithm 3. Algorithm

Algorithm 3 `F4` Algorithm

Input: A list of polynomials $F \in \mathbb{F}_q[x_1, \ldots, x_n]$ and a selection function SEL from a list of critical pairs to another list of critical pairs.
Output: Gröbner basis G of $\langle f_1, \ldots, f_k \rangle$ with respect to GREVLEX .

1: $G := F, \tilde{F_0^+} := F$ and $d := 0$
2: $P := \{\texttt{Pair}(f, g) | f, g \in G, f \neq g\}$
3: **while** $P \neq 0$ **do**
4: $\quad d := d + 1$
5: $\quad P_d := \text{SEL}(P)$
6: $\quad P := P \backslash P_d$
7: $\quad L_d := \texttt{Left}(P_d) \cup \texttt{Right}(P_d)$
8: $\quad \tilde{F_d^+} := \texttt{Reduction}(L_d, G)$
9: $\quad$ **for** $h \in \tilde{F_d^+}$ **do**
10: $\quad\quad P := P \cup \{\texttt{Pair}(h, g) | g \in G\}$
11: $\quad\quad G \leftarrow G \cup \{h\}$
12: $\quad$ **end for**
13: **end while**
14: **return** G

3 uses another algorithm `Reduction` in line 8. This algorithm generalizes or extends the idea of dividing a polynomial by a list of polynomials to idea of dividing

a list of polynomials by another list of polynomials. Thus, unlike the Buchberger's algorithm, the `Reduction` algorithm proposes to reduce a set of critical pairs with respect to some polynomials by using the algorithm of `Symbolic Processing` (Algorithm 5). The `Reduction` is described in Algorithm 4.

Algorithm 4 `Reduction`

Input: Sets $L \subset \mathtt{T} \times \mathbb{F}_q[x_1, \ldots, x_n]$ and $G \subset \mathbb{F}_q[x_1, \ldots, x_n]$
Output: A finite subset of $\mathbb{F}_q[x]$.

1: $F :=$ `SymbolicPreprocessing`(L, G)
2: $\tilde{F} :=$ Row echelon form of F
3: $\tilde{F^+} := \{f \in \tilde{F} | \mathtt{LT}(f) \notin \mathtt{LT}(F)\}$
4: **return** $\tilde{F^+}$

Algorithm 5 `SymbolicPreprocessing`

Input: Sets $L \subset \mathtt{T} \times \mathbb{F}_q[x_1, \ldots, x_n]$ and $G \subset \mathbb{F}_q[x_1, \ldots, x_n]$
Output: A finite subset of $\mathbb{F}_q[x]$.

1: $F := \{t \times f | (t, f) \in L\}$
2: Done := $\mathtt{LT}(F)$
3: **while** $\mathtt{T}(F) \neq$ Done **do**
4: Choose m an element of $\mathtt{T}(F) \backslash$Done
5: Done := Done $\cup \{m\}$
6: **if** m is top reducible modulo G **then**
7: $m = m' \times \mathtt{LT}(f)$ for some $f \in G$ and $m' \in \mathtt{T}$
8: $F := F \cup \{m' \times f\}$
9: **end if**
10: **end while**
11: **return** F

The Sel function applies the *normal strategy* for `F4` which outputs all the critical pairs from the set of critical pairs P, whose degree is the equal to the minimal degree amongst all such pairs [Fau99].

The algorithm terminates only when all the critical pairs have been processed. Faugère also proposed certain improvements on the `F4` algorithm, such as incorporating Buchberger's criterion (Theorem 2.3.21). `F4` uses this criterion to get rid of all such pairs which will not lead to a change in the Gröbner basis, i.e. those which have disjoint head pairs (Buchberger's first criterion) [BW98, Lemma 5.66]. This allows for making updates to the set of critical pairs as well as the basis elements which act as the divisors in the `Reduction` algorithm [BW98, Fau99]. An advantage of this strategy is that it does not require the input of a degree D, which is unlike Lazard's algorithm.

In Figure 2.1, we show a model run of the `F4` algorithm on `MAGMA`. We generate a generic system of 30 quadratic equations over 20 variables. From line 4 of Algorithm 2.3.2.3, we see that the Gröbner basis computation proceeds degree by degree, considering critical pairs in each degree from the set of critical pairs. In Figure 2.1, this degree is displayed by the "step degree" (see line 17,35,53). Each run of the while loop (line 3, Algorithm 3) computes the Gröbner basis for the corresponding step degree d. Another critical observation is the dimensions of the matrix that is considered at each step degree in the `Reduction` function. The number of columns of the matrix (see line 27,40, Figure 2.1) that the algorithm deals within each step degree is the total number of possible monomials up to the step degree. For example, in step degree 3, the total number of monomials up to degree 3 is given by

$$\binom{n+3-1}{3} + \binom{n+2-1}{2} + \binom{n}{1} + 1.$$

For $n = 20$ we obtain number of columns$= 1771$ and is easily verifiable from the Figure 2.1. While the number of columns is easy to determine and verify theoretically, the same cannot be said for the number of rows of the matrix. However, the log of the `F4` algorithms (i.e. Figure 2.1) gives the exact number of rows.

For example, in step degree 3, after symbolic processing of the list of polynomials the number of reductors (from the set L) and the number of reductees (given in G), it is observed that the number of rows is 629. Finally, after computing the row echelon form of this matrix of 629 rows and 1771 columns, we obtain 126 new polynomials that are added to the queue for the next while loop for degree 4.

The main advantage of using `F4` is the size of the matrices involved is much smaller than that considered by Lazard's Macaulay matrix-based algorithm. For the Macaulay matrix approach, at degree 3, the number of rows of the matrix before Gaussian elimination of the matrix is 1040. These come from rows of the form $x_i f_j$. Comparing this to the previous example, at step degree 3, the number of rows in the matrix is 629. Another aspect of `F4` which makes it more efficient than the previous methods, is the utilization of sparse linear algebra [Fau99]. During the computation of Gröbner basis, the systems that are encountered are in general sparse, consequently so are the Macaulay matrices involved. This is evident from the previous example provided in Figure 2.1 and 2.2. Observe as the step degree increases, the density of the matrices involved decreases. For example, at step degree 2 (Line 23,fig 2.1) the matrix has on average $\approx 50.26\%$ (116 out of 231 total columns) non zero columns per row, while at step degree 3 (Line 41), this reduces to $\approx 5.72\%$ (101 out of 1769 total columns). This decreases to $\approx 2.83\%$ and $\approx 3.06\%$ for step degree 5 and 6 respectively. Thus sparse matrix solving techniques give `F4` the upper hand in comparison to algorithms by Buchberger and Lazard from efficiency point of view .

```
********************
FAUGERE F4 ALGORITHM
********************
Coefficient ring: GF(2)
Rank: 20
Order: Graded Reverse Lexicographical
NEW hash table
Matrix kind: Packed GF(2)
Datum size: 0
No queue sort
Initial length: 30
Inhomogeneous
Initial queue setup time: 0.000
Initial queue length: 33
*******
STEP 1
Basis length: 30, queue length: 33, step degree: 2, num pairs: 24
Basis total mons: 3483, average length: 116.100
Number of pair polynomials: 24, at 231 column(s), 0.000
Average length for reductees: 116.71 [24], reductors: 113.67 [6]
Symbolic reduction time: 0.000, column sort time: 0.000
24 + 6 = 30 rows / 231 columns out of 231 (100.000%)
Density: 50.26% / 50.491% (116.1/r), total: 3483 (0.0MB)
Before ech memory: 32.1MB (=max)
Row sort time: 0.000
0.000 + 0.000 + 0.000 = 0.000 [24]
After ech memory: 32.1MB (=max)
Num new polynomials: 24 (100.0%), min deg: 2 [24], av deg: 2.0
Degree counts: 2:24
Queue insertion time: 0.000
New max step: 1, time: 0.000
Step 1 time: 0.000, [0.001], mat/total: 0.000/0.000, mem: 32.1MB
    (=max)
*******
STEP 2
Basis length: 54, queue length: 126, step degree: 3, num pairs:
    126
Basis total mons: 5838, average length: 108.111
Number of pair polynomials: 127, at 1314 column(s), 0.000
Average length for reductees: 99.30 [127], reductors: 101.72 [502]
Symbolic reduction time: 0.000, column sort time: 0.010
127 + 502 = 629 rows / 1769 columns out of 1771 (99.887%)
Density: 5.7225% / 10.325% (101.23/r), total: 63674 (0.2MB)
Before ech memory: 32.1MB (=max)
Row sort time: 0.000
0.000 + 0.000 + 0.000 = 0.000 [126]
After ech memory: 32.1MB (=max)
Num new polynomials: 126 (99.2%), min deg: 3 [126], av deg: 3.0
Degree counts: 3:126
Queue insertion time: 0.010
New max step: 2, time: 0.020
Step 2 time: 0.020, [0.017], mat/total: 0.000/0.020, mem: 32.1MB
    (=max)
*******
STEP 3
Basis length: 180, queue length: 1279, step degree: 4, num pairs:
    1231
```

Figure 2.1 – A snippet of the F4 algorithm on MAGMA: Part 1

```
STEP 4
Basis length: 724, queue length: 9760, step degree: 5, num pairs: 7487
Basis total mons: 1203298, average length: 1662.014
48 pairs eliminated
Number of pair polynomials: 7439, at 33617 column(s), 0.779
Average length for reductees: 2069.68 [7439], reductors: 591.57 [22472]
Symbolic reduction time: 0.480, column sort time: 0.019
7439 + 22472 = 29911 rows / 33854 columns out of 53130 (63.719%)
Density: 2.8333% / 4.69% (959.18/r), total: 28690066 (109.4MB)
Before ech memory: 160.2MB, max: 224.3MB
Row sort time: 0.009
2.440 + 0.009 + 0.799 = 3.250 [2247]
After ech memory: 192.2MB, max: 224.3MB
Num new polynomials: 2247 (30.2%), min deg: 5 [2247], av deg: 5.0
Degree counts: 5:2247
Queue insertion time: 3.129
New max step: 4, time: 7.670
Step 4 time: 7.669, [7.672], mat/total: 3.370/8.049, mem: 224.3MB (=max)
*******
STEP 5
Basis length: 2971, queue length: 56506, step degree: 6, num pairs: 38896
Basis total mons: 11468810, average length: 3860.252
1813 pairs eliminated
Number of pair polynomials: 37083, at 80239 column(s), 14.160
Average length for reductees: 4568.45 [37083], reductors: 1253.69 [64345]
Symbolic reduction time: 4.579, column sort time: 0.050
37083 + 64345 = 101428 rows / 80498 columns out of 230230 (34.964%)
Density: 3.0629% / 4.6033% (2465.6/r), total: 250080706 (954.0MB)
Before ech memory: 1345.4MB (=max)
Row sort time: 0.030
51.239 + 0.000 + 16.090 = 67.349 [8278]
After ech memory: 1450.8MB (=max)
Num new polynomials: 8278 (22.3%), min deg: 5 [1260], av deg: 5.8
Degree counts: 5:1260 6:7018
Queue insertion time: 53.839
New max step: 5, time: 140.020
Step 5 time: 140.019, [140.101], mat/total: 70.750/148.069, mem: 1452.0MB (=max)
```

Figure 2.2 – A snippet of the F4 algorithm: Part 2

Another interesting observation is that amount of time spent by individual subprocesses inside the F4 algorithm process. The majority time for each individual while loop is spent in computing the row echelon form of the matrix, which was discussed previously. For example, let us consider the Step 4 (line 1) and Step 5 (linen 20) from Figure 2.2. Step 4 takes a total time of 7.669 seconds (line 18). However, a majority amount of time (3.25 seconds) is spent inside the Reduction process. Interestingly, an equivalent amount of time is spent by the queue insertion procedure. This is because the implementation of F4 on MAGMA also implements an additional procedure of updating the set of critical pairs and reductors by using the Buchberger criterion. Step 5 in Figure 2.2 we observe that selecting the number of critical pairs at step degree 6 takes 14.16 seconds. Once the critical pairs are shortlisted, the algorithm proceeds to symbolically processing the list of pairs which can be expected to reduce down to zero later taking 4.6 seconds. Finally once the matrix of polynomials is obtained, computing the row echelon form takes 67.35 seconds.

2.3.2.4 Faugère's F5 Algorithm

In the previous sections, we looked at some of the algorithms for computing Gröbner basis. Even though these are efficient algorithms however, these algorithms share a common aspect. Recalling Buchberger's algorithm, computing the remainder of the S-polynomial division is redundant when the remainder is zero. Similarly in Lazard's algorithm, it is seen that a large fraction of the polynomials that are constructed reduces to zero. Thus a large fraction of the running time of these algorithms is spent in computing and reducing these polynomials which are discarded. Hence this part of the process is redundant.

From the Gröbner basis point of view, when an algorithm considers polynomials, which after performing some linear algebra operations reduces to zero, we say that the algorithm performs *reductions to zero*. These "reductions to zero" are also known as *syzygies*. The practical efficiency of Gröbner basis algorithms are greatly impacted by such reductions to zero [Fau02, Table 9]. Thus it is important to avoid them as much as we can. There are several ways to predict such reductions to zero [GM88, Fau02, EF14]. We shall focus on one such method, the F5 criterion [Fau02].

Assume we have two polynomials f_1, f_2 in the commutative polynomial ring $\mathbb{F}[x]$ of degrees d_1 and d_2. Constructing the matrix which contains all the multiples up to degree $d_1 + d_2$, the relation $f_1 f_2 - f_2 f_1 = 0$ induces reduction to zero during the linear algebra. These special reductions to zero are called *trivial syzygies*. The idea of the F5 criterion is to use the index of the rows of the Macaulay matrix to

prevent these reductions.

Proposition 2.3.25 (Matrix-F5 Criterion). *[Fau02] Given a system $F := (f_1, \ldots, f_m) \in \mathbb{F}_q^m[x_1, \ldots, x_n]$ of homogeneous polynomials of degree $d_1, \ldots, d_m$ respectively. Let a row in $\mathcal{M}_{\succ,d}(F)$ be indexed by (f_i, x^α), such that $x^\alpha \in \mathtt{LM}_\succ(f_1, \ldots, f_{i-1})$. Then this row is a linear combination of the rows in $\mathcal{M}_{\succ,d}(F)$ with smaller index, see Def 2.3.22.*

Algorithm 6 `Matrix-F5` Algorithm

Input: A list of polynomials $(f_1, \ldots, f_m) \in \mathbb{F}_q^m[x_1, \ldots, x_n]$ of degrees $d_1, \ldots, d_m$, a monomial ordering $\succ$ and a degree D.
Output: Gröbner basis G of $\langle f_1, \ldots, f_k \rangle$ with respect to $\succ$.

$G_{0,0}, \ldots, G_{0,D} \leftarrow \{\}, \ldots, \{\}$.
for d **from** 1 **to** D **do**
  for i **from** 1 **to** m **do**
    $\mathcal{M}_{\succ,d}(f_1, \ldots, f_i) \leftarrow$ Macaulay matrix for F w.r.t $\succ$ at degree d
    $\mathcal{M}^{\mathtt{F5}}_{\succ,d}(f_1, \ldots, f_i) \leftarrow$ Empty matrix
    for each (f_j, x^α) index of $\mathcal{M}_{\succ,d}(f_1, \ldots, f_i)$ **do**
      if $(d_j > d)$ **or** for all $g \in F_{j-1.d-d_j}, \mathtt{LM}_g \not| \, x^\alpha$ **then**
        Add to $\mathcal{M}^{\mathtt{F5}}_{\succ,d}(f_1, \ldots, f_i)$ the row with index (f_j, x^α)
      end if
    end for
    $\widetilde{\mathcal{M}_{\succ,d}(F)} \leftarrow$ Reduced echelon form of $\mathcal{M}^{\mathtt{F5}}_{\succ,d}(F)$
    $P_{i,d} \leftarrow$ Non-zero polynomials represented by the rows of $\widetilde{\mathcal{M}_{\succ,d}(F)}$
    $G_{i,d} \leftarrow G_{i,d-1} \cup \{g \in P_{i,d} : (\forall h \in G_{i.,d-1})\mathtt{LM}_\succ(h) \not| \, \mathtt{LM}_\succ(g)\}$
  end for

end for
return $G_{r,D}$

2.3.3 Complexity of Gröbner Basis Computation

Computing Gröbner basis is at least as hard as solving a system of polynomials. The worst case time complexity of Gröbner basis methods is known to be doubly exponential in the number of variables, even for quadratic systems [MM82]. This was achieved for a positive dimensional system where the complexity is precisely the $2^{2^{n/10}}$, where n is the number of variables. However, in cryptography we deal with systems which are zero-dimensional. Hence we do not encounter such doubly exponential complexity of computing Gröbner basis. For instance, given a regular (see Definition 2.3.28) and square polynomial system with degrees $(d_1, \ldots, d_n)$

having only a finite number of roots, then computing its GRevLex Gröbner basis has a time complexity which is polynomial in $\prod_i^n d_i$ (the Bézout's bound) [Laz83, FGHR13].Specifically for Degree reverse lexicographic ordering (GrevLex), the highest degree of the Gröbner basis elements is bounded by

$$\text{The Macaulay Bound: } 1 + \sum_{i=1}^{n}(d_i - 1).$$

Revisiting the idea of using Macaulay matrices to compute Gröbner bases, we know that a Macaulay matrix $\mathcal{M}_{\succ,d}$ at degree d, has

$$C_d := \binom{n+d-1}{d}, \quad R_d := |\ell_{d-d_1}| + \cdots + |\ell_{d-d_m}|,$$

(where ℓ is as defined in Definition 2.3.22) where C_d and R_d represents the number of columns and rows respectively. A basis of the rows is obtained by computing the reduced row echelon form of the matrix $\mathcal{M}_{\succ,d}$. Fast matrix multiplication techniques proposed in [Sto00] provides a way to compute this in complexity $\mathcal{O}(R_d C_d r^{\omega-1})$ where r is the rank of the matrix after the reduction. Now, the number of rows R_d is upper bounded by mC_d and the rank is upper bounded by C_d. Thus, the total complexity of doing Gaussian reduction is bounded by $\mathcal{O}(mC_d^{\omega})$. Now this process is performed for each step of the degrees from $\min(d_1, \ldots, d_m), \ldots, d_{max}$, where d_{max} is the maximal degree up to which Gröbner basis computation continues. Hence we have the following upper bound on the complexity of Gröbner basis computation.

Proposition 2.3.26. *Let $f_1, \ldots, f_m$ be a system of homogeneous polynomials in $\mathbb{F}_q[x_1, \ldots, x_n]$. The number of operation required to compute the Gröbner basis of the ideal $I_{d_{max}} := \langle f_1, \ldots, f_m \rangle_D$ for some graded monomial ordering and a degree bound D is given by*

$$\mathcal{O}\left(mD\binom{n+D-1}{D}^{\omega}\right),$$

where $2 \leq \omega \leq 3$ is the exponent of matrix multiplication over $\mathbb{F}_q$ [Wil12].

Example 2.3.27. *Le us revisit the example 2.2.1, where we have 90 quadratic equations in 80 variables. Assuming that the degree D to be the Macaulay bound, we have $D := 81$, the complexity of computing the* GRevLex *Gröbner basis has a complexity approximately 2^{325} bit operations.*

Note 2.3.1. *It is worth mentioning here that the computation of Gröbner basis is generally more efficient for some monomial ordering than others. Typically, for the* GRevLex *order Gröbner basis computation is more efficient while* Lex *ordering being the least efficient.* Lex *ordering provides the easiest way of recovering all the common zeros of the system, thanks to the shape position formation. To change a*

Gröbner basis from a GREVLEX *ordering to* LEX *ordering, the best known algorithm is* `FGLM`*, which was proposed by Faugère et al. [FGLM94]. To recover the Gröbner basis for* LEX *ordering from another graded monomial ordering* GREVLEX*, the complexity of algorithm FGLM for computing the Gröbner basis of a zero-dimensional ideal with degree D is $\mathcal{O}(nD^\omega)$ where ω is the linear algebra constant.*

This Macaulay bound may be the highest possible degree for the elements that might occur in the Gröbner basis, however, in the example provided (90 quadratic equations over 80 variables), the number of equations is more than the number of variables. Hence in this case (and practically for most multivariate encryption, as well shall see later), for the highest degree occurring in the Gröbner basis computation, the Macaulay bound is not sharp. Thus, this complexity of computing the Gröbner basis is not optimal. To improve on this complexity, we need to compute a tighter upper bound of D and for that reason, we look into the notion of "regular" and "semi-regular" systems for which the bound on the degree is quite well defined and is much sharper than the Macaulay bound.

Definition 2.3.28 (Regular Sequence)**.** *A sequence $f_1, \ldots, f_m \in \mathbb{F}_q[x_1, \ldots, x_n]$ of non-zero homogeneous polynomials (where $m \leq n$) is called a regular sequence if forall $i \in \{1, .., m\}$ and forall $g \in \mathbb{F}[x_1, \ldots, x_n]$*

$$gf_i \in \langle f_1, \ldots, f_{i-1} \rangle \implies g \in \langle f_1, \ldots, f_{i-1} \rangle.$$

Proposition 2.3.29. *[BFS15, Thm 9.] If $(f_1, \ldots, f_m)$ is a regular sequence, then the* `Matrix-F5` *algorithm performs no reduction to zero.*

Definition 2.3.30 (Hilbert Function and Series)**.** *Given a homogeneous ideal $I \subset \mathbb{F}_q[x_1, \ldots, x_n]$, we denote the the set of homogeneous polynomials (along with the zero polynomial) of total degree d as $\mathbb{F}_q[x_1, \ldots, x_n]_d$. We consider the subset $I_d = I \cap \mathbb{F}_q[x_1, \ldots, x_n]_d$. This I_d is a vector subspace of $\mathbb{F}_q[x_1, \ldots, x_n]_d$. The Hilbert function $\mathtt{HF}_I$ of I is defined as*

$$\mathtt{HF}_I(d) = \dim(\mathbb{F}_q[x_1, \ldots, x_n]_d \backslash I_d).$$

The Hilbert Series H_I is the generated series of the Hilbert function

$$H_I(t) = \sum_{i=0}^{\infty} \mathtt{HF}_I(i) t^i.$$

This series is a power series which can be written as

$$H_I(t) = \frac{P(t)}{(1-t)^d},$$

where $P(t)$ is a polynomial and d is the dimension of the ideal I.

Property 2.3.31. *[BFS04]*

- *A sequence of homogeneous polynomials* $(f_1, \ldots, f_m) \in \mathbb{F}_q[x_1, \ldots, x_n]$*, of degrees* $d_1, \ldots, d_m$*, is regular if and only if its Hilbert series*

$$H(t) := \frac{\prod_{i=1}^{m}(1-t^{d_i})}{(1-t)^n},$$

- *The highest degree of elements of a Gröbner basis for the* GREVLEX *ordering is less than the Macaulay bound,i.e.*

$$\sum_{i=1}^{n}(d_i - 1) + 1.$$

When the number of polynomials is greater than the number of variables (i.e. $m > n$), this notion of regular system doesn't hold. However, a weak notion of regular system can be introduced.

Definition 2.3.32 (d-regular system)**.** *[BFS04] A zero-dimensional over-determined* $(m \geq n)$ *sequence of homogeneous equations* $(f_1, \ldots, f_m)$ *is called* d*-regular, if for all* $i = 1, \ldots, m$ *there exists a* g *such that*

$$deg(g) < d - d_i \;\; \textit{and} \;\; gf_i \in \langle f_1, \ldots, f_{i-1}\rangle,$$

then $g \in \langle f_1, \ldots, f_{i-1}\rangle$*.*

Definition 2.3.33 (Degree of regularity)**.** *For a zero-dimensional ideal* $I := \langle f_1, \ldots, f_m\rangle$ $(m \geq n)$*, the degree of regularity is given by*

$$D_{reg} = \min\left\{d \geq 0 \mid dim_{\mathbb{F}_q}(\{f \in I, \;\; deg(f) = d\}) = \binom{n+d-1}{d}\right\}.$$

Thus the monomials in degree D_{reg} are the leading terms for the elements in the ideal. Thus this value is clearly an upper bound on the degree of the elements occurring in the Gröbner basis for any monomial ordering.

Remark 2.3.34. *The degree of regularity for regular sequences* $(m \leq n)$ *is given by the Macaulay bound* $\sum_{i=0}^{m}(d_i - 1) + 1$ *[Laz83, Mac02].*

Definition 2.3.35 (Semi-regular system)**.** *[BFS04] If a system of equations is* D_{reg}*-regular, then it is also called as semi-regular system.*

From the Definition 2.3.33 we can compute the degree of regularity, however, the following proposition gives another more efficient quantitative approach to compute this degree of regularity for a system of semi-regular system.

Proposition 2.3.36. *A system of equations $(f_1, \ldots, f_m) \in \mathbb{F}_q^m[x_1, \ldots, x_n]$ of respective degrees $(d_1, \ldots, d_m)$ is semi-regular if and only if the degree of regularity is given by index of the first non-positive coefficient of the series*

$$\mathcal{H}(t) = \frac{\prod_{i=1}^{m}(1-t^{d_i})}{(1-t)^n}, \quad \text{when } m > n \text{ and } q > 2.$$

When we restrict the solutions to the system of equations to the finite field $\mathbb{F}_q$, we add field equations $x_i^q - x_i$ to the ideal. In that case, the degree of regularity is given by the index of first non-positive coefficient of the series

$$H(t) := \frac{(1-t^q)^n}{(1-t)^n} \prod_{i=1}^{m} \left(\frac{1-t^{d_i}}{1-t^{qd_i}} \right).$$

In particular, when $q = 2$, the Hilbert series is

$$H(t) := \frac{(1+t)^n}{\prod_{i=1}^{m}(1+t^{d_i})}.$$

Example 2.3.37. *Under the semi-regularity assumption on a system of 90 quadratic equations in $\mathbb{F}_2[x_1, \ldots, x_n]$ over 80 variables, the Hilbert series is given by*

$$\mathcal{H}(t) := \frac{(1+t)^{80}}{(1+t^2)^{90}},$$

$$\mathcal{H}(t) := 1 + 80\,t + 3070\,t^2 + 74960\,t^3 + 1301275\,t^4 + 16973216\,t^5 + 170972620\,t^6 +$$
$$1339513760\,t^7 + 8025176185\,t^8 + 34355612720\,t^9 + 78718740802\,t^{10} -$$
$$168147088720\,t^{11} - 2525289305045\,t^{12} - \cdots .$$

The index of the first non positive coefficient in this series is 11. Thus the the degree of regularity, for this case is 11. *This maximal degree occurring in computation, predicted under the semi-regular assumption is a much sharper upper bound than the Macaulay bound as considered in example 2.3.27. Hence for computing the Gröbner basis for this case, one needs at approximately 2^{100} bit operations.*

M. Bardet, in his PhD thesis [Bar04], provided the asymptotic estimates of the degree of regularity for a system of semi-regular sequence of equations. In particular, for quadratic equations, [Bar04] we have the following estimate for the degree of regularity.

Proposition 2.3.38. *[Bar04, Prop 4.1.4] The degree of regularity of a semi-regular sequence of αn homogeneous quadratic polynomials in n variables, where $\alpha \geq 1$ is a constant, behaves asymptotically as*

$$D_{reg} \sim \left(\alpha - \frac{1}{2} - \sqrt{\alpha(\alpha-1)} \right) n + \mathcal{O}(n^{1/3}), \quad n \to \infty.$$

Example 2.3.39. *We continuing with the same example of* 90 *homogeneous quadratic polynomials in* 80 *variables. Asymptotically, using Proposition 2.3.38, the estimated degree of regularity for such a system is approximately 20. One should note that this is an asymptotic estimate, hence one cannot expect this to be sharper upper bound for the degree of regularity than the one computed in Example 2.3.37, where the degree of regularity (which was 11) was computed explicitly from the Hilbert series.*

Definition 2.3.40 (Affine degree of regularity)**.** *Let* $(f_1, \ldots, f_m) \in \mathbb{F}_q[x_1, \ldots, x_n]$ *be a sequence of affine polynomials and* $I := \langle f_1, \ldots, f_m \rangle$*. For each polynomial* f_i*, let* f_i^h *denote the homogeneous part of* f_i *of the largest degree. The sequence* $f_1, \ldots, f_m$ *are said to be semi-regular if the homogeneous sequence* $f_1^h, \ldots, f_m^h$ *is semi-regular over* $\mathbb{F}_q$*. We define the degree of regularity of this affine ideal* I *to be the degree of regularity of the homogeneous ideal* $I^h := \langle f_1^h, \ldots, f_m^h \rangle$*, i.e.* $D_{reg}(I) = D_{reg}(I^h)$*.*

2.4 Hybrid Combinatorial-Algebraic methods

2.4.1 Classical Hybrid Algorithms

In this section, we present two classical hybrid algorithms to solve the $\mathtt{PoSSo}_q$ problem. The first method was proposed by Luk Bettale *et al.* [BFP09, BFP12] which mixes exhaustive search and Gröbner basis technique. The core idea is to fix a certain fraction of the variables. Thus given a system of m polynomial equation in $\mathbb{F}_q[x_1, \ldots, x_n]$ over n variables, one fixes $k < n$ variables to obtain m new polynomial equations over $n - k$ variables. Now instead of computing one single Gröbner basis, one computes q^k subsystems over $n - k$ variables. Choosing an appropriate value of k, which gives us a gain in the complexity by computing Gröbner basis over a lesser number of variables allows us to overcome the loss of doing an exhaustive search. This is the main rationale of this approach.

Proposition 2.4.1. *[BFP09, BFP12] Let* $\mathbb{F}_q$ *be a finite field and* $(f_1, \ldots, f_m) \in \mathbb{F}_q^m[x_1, \ldots, x_n]$ *be polynomial system of degree* d*. Let* D_{reg} *be the maximal degree of regularity of all the systems*

$$\left\{ \left\{ f_1(x_1, \ldots, x_{n-k}, v_1, \ldots, v_k), \ldots, f_m(x_1, \ldots, x_{n-k}, v_1, \ldots, v_k) \right\} : (v_1, \ldots, v_k) \in \mathbb{F}_q^k \right\}.$$

If the systems are zero-dimensional, the complexity of the hybrid approach, C_{hyb}*, is bounded from above by*

$$\mathcal{O}\left(\min_{0 \leq k \leq n} \left(q^k \cdot \left(m \cdot \binom{n - k + D_{reg} - 1}{D_{reg}}^{\omega} \right) \right) \right).$$

where d is the maximal degree of the ideal (i.e. number of solutions counted with multiplicities in the algebraic closure of $\mathbb{F}_q$) generated by the each q^k subsystem and $2 \leq \omega \leq 3$ is a linear algebra constant.

Proposition 2.4.1 does not give any method to compute the best trade-off for the number of fixed variables, k, such that the gain in the hybrid Gröbner basis algorithm can be maximized with respect to generic Gröbner basis algorithms. However, for a semi-regular system (see Definition 2.3.35), knowing sharp estimates of the degree of regularity allows us to directly compute this trade-off. As seen earlier, Proposition 2.3.38 gives us an asymptotic estimate to the degree of regularity. Powered with this asymptotic estimate the idea is to find the value of k for which the C_{hyb} has a global minimum. The value of the k depends on m and q. Putting it more formally,

Theorem 2.4.2. *[BFP09, BFP12] Let $F = (f_1, \ldots, f_m) \subset \mathbb{F}_q[x_1, \ldots, x_n]$ be a semi-regular sequence of quadratic equations with $m = \alpha n$ ($\alpha > 0$) and where for all values of $0 \leq k \leq n$ and all possible vectors $(v_1, \ldots, v_k) \in \mathbb{F}_q^k$, the quadratic equations*

$$f_1(x_1, x_{n-k}, v_1, \ldots, v_k), \ldots, f_m(x_1, x_{n-k}, v_1, \ldots, v_k),$$

are semi-regular. Let $A(\beta)$

$$\begin{aligned} A(\beta) = \log q &+ \omega(\log n + \log(1-\beta)) \\ &- \frac{\omega}{2}\left(1 + \sqrt{\frac{\alpha}{\alpha+\beta-1}}\right)\left(\log n + \log\left(\alpha + \frac{1-\beta}{2} - \sqrt{\alpha(\alpha+\beta-1)}\right)\right) \\ &- \frac{\omega}{2}\left(1 - \sqrt{\frac{\alpha}{\alpha+\beta-1}}\right)\left(\log n + \log\left(\alpha - \frac{1-\beta}{2} - \sqrt{\alpha(\alpha+\beta-1)}\right)\right), \end{aligned}$$

and let β_0 be a non negative real root of $A(\beta)$. The best trade off is to fix $k = \lfloor \beta_0 n \rfloor$ variables.

The values of this root β_0 can be computed explicitly (see Table 4.1 of [Bet11]). For example, over a finite field of $\mathbb{F}_2$, with $2n$ quadratic equations over n variables, fixing $k = \lfloor 0.51n \rfloor$ variables gives the best complexity for the hybrid approach. However, with the same number of equations but over a finite field $\mathbb{F}_{2^2}$, the most optimal value of k is $\lfloor 0.042n \rfloor$.

Example 2.4.3. *We again revisit the example 2.2.1, where we have 90 quadratic equations over 80 variables in $\mathbb{F}_2[x_1, \ldots, x_n]$. According to [Bet11], in this case we have $k = \lfloor 0.7n \rfloor$. The degree of regularity of the subsystems is bounded by index of the first non positive coefficient of the Hilbert series*

$$\mathcal{H}(t) := \frac{(1+t)^{24}}{(1+t^2)^{90}},$$

which turn out to be 3. Thus by Proposition 2.4.1, the complexity of the hybrid Gröbner basis in this case is 2^{87} bit operations. The following Table 2.1 lists the complexity of the Gröbner basis computation for one subsystem, the hybrid complexity for k subsystems and the degree of regularity that is observed while computing the solutions to the system of equations for various values of k.

k	GB	Hybrid	Dreg
52	38.37	90.37	4
54	37.56	91.56	4
56	**30.76**	**86.76**	**3**
58	30.04	88.04	3
60	29.25	89.25	3

Table 2.1 – Example 2.4.5. The GB and Hybrid columns implies that the number of bit operations is 2^b, where b is the column entry. The complexity of the Hybrid column is the addition of the entries in column k and GB.

In 2013, a second algorithm was proposed by Bardet *et al.* [BFSS13] in order to solve the $\mathtt{MQ}_2$ problem in specific. They proposed a deterministic algorithm `BooleanSolve` (and its probabilistic variant) which fixes k variables to all possible values and then checks the consistency of the new over-determined system of equations. Given a system of Boolean quadratic equations $(f_1, \ldots, f_m) \in \mathbb{F}_2^m[x_1 \ldots, x_n]$, one can search for polynomials $h_1, \ldots, h_{m+n-k} \in \mathbb{F}_2[x_1, \ldots, x_{n-k}] \subset \mathbb{F}_2[x_1 \ldots, x_n]$ in variables $x_1, \ldots, x_{n-k}$ such that

$$h_1\bar{f}_1 + \cdots + h_m\bar{f}_m + h_{m+1}x_1(1-x_1) + \cdots + h_{m+n-k}x_{n-k}(1-x_{n-k}) = 1 \quad (2.1)$$

where $(\bar{f}_i)$ are derived equations from f_i after fixing k variables. Given a degree bound D, existence of such polynomials can be easily checked using linear algebra. Satisfaction of the above equation implies that the input system is inconsistent. This follows directly from Hilbert Nullstellensatz theorem [CLO06], which states that a system is inconsistent if and only if 1 belongs to the ideal generated by the polynomials.

Like the hybrid approach of [Bet11], this approach also demands a trade off for degree bound D and the number of fixed variables k. Specifically for the case of Boolean system under certain hypothesis [BFSS13, Proposition 6], the degree D is bounded by the degree of the polynomial [BFSS13, Corollary 1]

$$\mathtt{HS}_{n,m}(t) = \frac{(1+t)^{n-k}}{(1-t)(1+t^2)^m}.$$

The method to compute the value of k is mostly dependent on the complexity of the linear algebra stage of finding the polynomials h_i's. Hence the approach taken

here is quite similar to that of [Bet11], in order to optimize the overall cost. In the following theorem [BFSS13] give us a complexity estimate of the algorithm in terms of the ratio of the number of equations to the number of variables and n, such that the complexity of an exhaustive search is minimum with the choice of k.

Theorem 2.4.4. *[BFSS13, Theorem 2] Let $f_1, \ldots, f_m$ be a system of quadratic polynomials in $\mathbb{F}_2[x_1, \ldots, x_n]$ with $m = \lceil \alpha n \rceil$ and $\alpha \geq 1$. Then deterministic variant of `BooleanSolve` finds all the roots in $\mathbb{F}_2^n$ with the number of arithmetic operations in $\mathbb{F}_2$ in*

$$\mathcal{O}(2^{(1-0.112\alpha)n}),$$

if the system is semi-regular and $k = (1 - 0.27\alpha)n$. The number of expected operations for the Las-Vegas variant (probabilistic) of `BooleanSolve` is bounded by

$$\mathcal{O}(2^{(1-0.208\alpha)n}),$$

taking an optimal number of $k = (1 - 0.55\alpha)n$ fixed variables.

The probabilistic variant of `BooleanSolve` behaves like the deterministic variant with the choice of linear algebra constant is $\omega = 2$, i.e. the linear algebra step of the algorithm performs in quadratic complexity.

Example 2.4.5. *Using the method of [BFSS13], for 90 quadratic equations and 80 variables, the deterministic `BooleanSolve` algorithm takes nearly 2^{70} bit operations and involves fixing 56 variables. This value of k is optimal from both the theoretical as well as practical point of view for `BooleanSolve`. The value of $k = 56$ for this specific example is backed by the idea that both, deterministic `BooleanSolve` and hybrid method of [BFP09] take similar approaches for computing the optimal k. Using the probabilistic variant of `BooleanSolve`, the complexity takes approximately 2^{62} bit operations with $k = 31$ and assuming the linear algebra $\omega = 2$.*

2.4.2 Quantum Hybrid Approach

In similar lines to the combinatorial approaches, [HRSS16] derives a quantum variant of hybrid approach from [Bet11, BFP12] by explicitly using Grover's algorithm [Gro96] to accelerate the exhaustive search over the fixed variables. However, they do not provide any asymptotic complexity of this approach. [FHK+17] goes a step further, firstly it builds a new (inspired) algorithm called `QuantumBooleanSolve` on top of the state-of-the-art `BooleanSolve` algorithm [BFSS13] and secondly, proposes an asymptotic complexity of `QuantumBooleanSolve`. This algorithm is different from the one proposed in [HRSS16]. In `QuantumBooleanSolve`, the authors instantiate a non-trivial quantum oracle which has a specific quantum circuit required for simplified Gröbner basis computation using Macaulay matrices. In this work, our main interest lies in the complexity of such algorithms and thus the

algorithms themselves have been discussed. However, for more detailed working of the `QuantumBooleanSolve` we invite the reader to have a look at [FHK+17].

Theorem 2.4.6. *[FHK+17, Theorem 1] There is a quantum algorithm which solves the* $\texttt{MQ}_2$ *problem and requires*

- *evaluation of* $\mathcal{O}(2^{0.47n})$ *quantum gates for the deterministic variant,*
- *evaluation of expected* $\mathcal{O}(2^{0.462n})$ *quantum gates for the probabilistic version.*

This algorithm beats the Quantum exhaustive search (see 2.2) which is $\mathcal{O}(2^{n/2})$.

Example 2.4.7. *Continuing with the same example as before, the deterministic* `QuantumBooleanSolve` *algorithm solve the system of equations requires the evaluation of at least* $\approx 2^{38}$ *quantum gates, which is much faster than the Quantum exhaustive search which requires the evaluation of minimum* 2^{59} *quantum gates.*

Example 2.4.8. *Let us again take the example of 90 quadratic equations over 80 variables in* $\mathbb{F}_2[x_1, \ldots, x_n]$. *Using Grover's approach along with hybrid Gröbner basis technique [BFP09], we can estimate that this process takes* 2^{59} *arithmetic operations.*

2.5 Conclusion

In this chapter, we discussed the `NP`-Hard problem of `PoSSo`. We also describe various algorithms for solving an instance of the `PoSSo` problem, broadly classifying into three types: combinatorial-based, algebraic algorithms and a combination of these which we call as hybrid algorithms. We also describe some state-of-the-art quantum algorithms which solve instances of this problem. In this following chapter, we shall discuss one application area of the `PoSSo` problem, namely Multivariate cryptography.

Chapter 3

Quantum-Safe Public-key Cryptography

Abstract

Public-key cryptography is based on certain problems that are known to be hard to solve. Unfortunately, with the advent of quantum computers, some of these hard problems are solvable. Although, a subset of such problems are known to be still safe from quantum attacks. Thus, quantum-safe public-key cryptography has become major field of interest for security researchers and is exhibited by the strong steps taken by `NIST` to standardize new quantum-safe cryptosystems. In this chapter, we present some historical as well as current cryptosystems that are quantum safe. In particular, we focus on multivariate based cryptography which is also the main focus of this thesis.

In the following sections we take special interest in one category of post-quantum cryptosystems: Multivariate based cryptography. We start with discussions on some of the most prominent state-of-the-art multivariate cryptosystems and also list the known attacks on such systems in some detail. The final part of the chapter discusses lattice based cryptography briefly, where we list one of the main lattice based hard problems that forms the security foundation of lattice based cryptosystems. In addition, we also present a lattice based scheme: `Frodo`, that influenced the design of a multivariate key-exchange scheme by us, the details of which have been discussed in great detail in Chapter 6.

3.1 Multivariate Public-Key Cryptography

Multivariate Public-Key Cryptography (`MPKC`) is a subclass of asymmetric cryptography that deals with the cryptographic schemes whose hardness are based on the $\texttt{PoSSo}_q$ problem (see Section 2.1). `MPKC` is a very active area of research

that culminated with the NIST Post-Quantum Cryptography standardization process [CCJ+16]. As mentioned previously in Chapter 1, it is one of the candidate areas for design of post-quantum cryptosystems. About 10% out of 69 submissions were multivariate. In the following section, we give an overview of the most prominent multivariate schemes.

3.1.1 General Structure

A special subclass of MPKC is Multivariate Quadratic (MQ) cryptography, that relies on the hardness of solving the MQ problem (see Chapter 2). The design principle of a MQ cryptosystem can be traced back to the first ever known example, C*, that was proposed by Matsumoto Imai [MI88]. Before we discuss the public and the private keys, we give a few prerequisites.

Definition 3.1.1. *Let $\mathbb{E}$ be a simple extension field of $\mathbb{F}_q$ of degree n, $\omega \in \mathbb{E}$ be the primitive element $\mathbb{E}/\mathbb{F}_q$ and let $\mathbb{F}_q^n$ be the corresponding vector space. Then $\mathbb{E}$ can be considered as a vector space over $\mathbb{F}_q^n$ through a vector space isomorphism $\phi : \mathbb{E} = \mathbb{F}_{q^n} \to \mathbb{F}_q^n$ such that for any element $v = v_1 + v_2\omega + \cdots + v_n\omega^{n-1} \in \mathbb{E}$, we have*

$$\phi(v) = (v_1, v_2, \ldots, v_n) \in \mathbb{F}_q^n.$$

With these definitions we can now enlist the secret-key of the C* scheme. It includes a function $F : \mathbb{F}_q^n \mapsto \mathbb{F}_q^n$ such that $\forall\, w \in \mathbb{F}_q^n$ we have,

$$F(w) = \phi \circ (\phi^{-1}(w))^{1+q^\alpha},$$

where ϕ is a map from $\mathbb{F}_{q^n}$ to $\mathbb{F}_q^n$ (see Definition 3.1.1). The choice of $\alpha \in \mathbb{N}$ is such that $\gcd(1 + q^\alpha, q^n - 1) = 1$. The polynomials resulting from F are of degree 2 and have a certain structure. Therefore, it becomes imperative to mask the system of polynomials. We define $\mathrm{Aff}_n(\mathbb{F}_q) \simeq \mathbb{F}_q^{n\times n} \times \mathbb{F}_q^n$ as the collection of the invertible affine transformations over $\mathbb{F}_q^n$. The secret-key also comprises of two transformations $\mathcal{S} = (S, \underline{s}) \in \mathrm{Aff}_n(\mathbb{F}_q)$ and $\mathcal{T} = (T, \underline{t}) \in \mathrm{Aff}_n(\mathbb{F}_q)$ where n is a positive integer.

The public-key is given by the construction $P = \mathcal{T} \circ F \circ \mathcal{S}$ where $\circ$ function composes two maps. Thus public-key is a system of polynomials $(p_1, \ldots, p_n) \in \mathbb{F}_q^n[x_1, \ldots, x_n]$ such that

$$(p_1, \ldots, p_n) = T \cdot (f_1(x'_1, \ldots, x'_n), \ldots, f_n(x'_1, \ldots, x'_n)) + \underline{t},$$

where $F = (f_1, \ldots, f_n) \in \mathbb{F}_q^n[x_1, \ldots, x_n]$ and $(x'_1, \ldots, x'_n) = S \cdot (x_1, \ldots, x_n) + \underline{s}$. The main idea is to mask the polynomial system F with the application of the affine transformations $\mathcal{S}$ and $\mathcal{T}$.

The number of public-key equations is equal to the number of variables in the case of $\mathtt{C}^*$, however, for other multivariate schemes, it is not always the case. Hence from now on, we shall denote the number of public-key equations as m and the number of variables as n.

Most current multivariate cryptosystems employ the same design principle as $\mathtt{C}^*$ to construct the public-key P: composing a m-tuple quadratic polynomials $F : \mathbb{F}_q^n \mapsto \mathbb{F}_q^m$, which we call as a *central trapdoor*, with two invertible affine transformations $\mathcal{S} \in \mathrm{Aff}_n(\mathbb{F}_q)$ and $T \in \mathrm{Aff}_m(\mathbb{F}_q)$. These schemes differ in the construction of the central map F as we shall see in Section 3.1.2. We point out that we can also use linear transformations S, T instead of the affine transformations $\mathcal{S}, \mathcal{T}$. This restriction may be used for ease of notation.

3.1.1.1 Encryption and Decryption

To encrypt a message $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{F}_q^n$, one computes the ciphertext $\mathbf{c} \in \mathbb{F}_q^m$ by evaluating the public-key $(p_1, \ldots, p_m) \in \mathbb{F}_q^m[x_1, \ldots, x_n]$ over the message $\mathbf{a}$, i.e.

$$\mathbf{c} = (c_1, \ldots, c_m) = (p_1(a_1, \ldots, a_n), \ldots, p_m(a_1, \ldots, a_n)) \in \mathbb{F}_q^m.$$

The number of operations required for encryption is effectively upper bounded by $\binom{n+2}{2}$ multiplications and $\binom{n+2}{2} - 1$ additions for each of the m quadratic polynomials in the public-key. Thus, overall we have $\mathcal{O}(mn^2)$ arithmetic operations over $\mathbb{F}_q$.

The holder of the secret-key can then decrypt the ciphertext $\mathbf{c} \in \mathbb{F}_q^m$ by inverting each polynomial map that composes the public-key. It should be noted that the central trapdoor F in general is not a bijection. Inverting the map F implies the application of the decryption procedure. For ease of notation, we denote this decryption process by F^{-1}. Hence we have

$$\mathbf{a}' = \mathcal{S}^{-1} \circ F^{-1} \circ \mathcal{T}^{-1}(\mathbf{c}) \in \mathbb{F}_q^n,$$

where $\circ$ is the composition map previously defined. For decryption to output a unique plaintext, one requires the public-key to be an injective map. Thus we need $m \geq n$. The number of operations for decryption depends on the choice of the trapdoor chosen for constructing the map F.

3.1.1.2 Signature

A digital signature can also be constructed using similar ideas. Digital signature involves two processes: signature generation and signature verification. To generate a signature $\mathbf{s} \in \mathbb{F}_q^n$ for a message $\mathbf{a} \in \mathbb{F}_q^m$, one can apply the decryption process to find a solution. However, unlike the decryption process in encryption schemes, signature generation need not be unique. Thus one can have $m < n$. So any $\mathbf{s} \in \mathbb{F}_q^n$ which is a solution to

$$\mathbf{s} = \mathcal{S}^{-1} \circ F^{-1} \circ \mathcal{T}^{-1}(\mathbf{a}),$$

is a possible signature to the message $\mathbf{a}$, where $\circ$ is the composition map. Signature verification is simply the process of evaluating the public-key $P = \mathcal{T} \circ F \circ \mathcal{S}$ over the signature $\mathbf{s}$. The signature $\mathbf{s}$ is a correct signature for the corresponding message $\mathbf{a}$ if $P(\mathbf{s}) = \mathbf{a}$.

3.1.2 Historical Cryptosystems

Amongst the current cryptosystems, the most significant multivariate primitive, which was proposed by Patarin, is *Hidden Field Equations* (HFE) [Pat96]. The central trapdoor of HFE is quite different from C*. For HFE, it is a degree bounded *extended Dembowski-Ostrom polynomial* [DO68, CM97] map

$$g : X \in \mathbb{F}_{q^n} \mapsto y = \sum_{0 \leq i,j \leq d} \alpha_{ij} X^{q^i+q^j} + \sum_{0 \leq i \leq d} \beta_i X^{q^i} + \gamma_0,$$

$$\text{where} \begin{cases} \alpha_{ij} X^{q^i+q^j} & \text{with } \alpha_{ij} \in \mathbb{F}_{q^n} \text{ are the quadratic terms,} \\ \beta_i X^{q^i} & \text{with } \beta_i \in \mathbb{F}_{q^n} \text{ are the linear terms,} \\ \gamma_0 & \text{with } \gamma_0 \in \mathbb{F}_{q^n} \text{ is the constant term,} \end{cases}$$

for $i, j \in \mathbb{N}$ and some degree $d \in \mathbb{N}$. Inverting g implies solving a univariate equation of high degree over $\mathbb{F}_{q^n}$. This inversion has been quite extensively studied in [Ber67, CZ81]. The expected cost of computing using the algorithm of [Ber67] is $\mathcal{O}(nD^2 \log q + D^3)$, where D is the maximal degree of g. For efficient inversion of g, the degree D should be small, which in turn implies d should be small.

In HFE, it is important to note that the central trapdoor is defined as a polynomial map over some extension field $\mathbb{F}_{q^n}$ and has a univariate representation. However, this polynomial also has an equivalent multivariate representation over the field $\mathbb{F}_q$.

Note 3.1.1. *Due to Frobenius automorphism, $X \to X^q$ is a linear mapping in the finite field $\mathbb{F}_q$ as well as the extension field $\mathbb{F}_{q^n}$. Therefore, all sums of monomials that are of the form $X^{q^i} \in \mathbb{F}_{q^n}[X]$, $0 \leq i < n$ are also linear over $\mathbb{F}_q[x_1, \ldots, x_n]$ [SK99], where $X = \phi^{-1}(x_1, \ldots, x_n)$.*

Lemma 3.1.2. *Let $\mathbb{F}_q$ be the base field and $\mathbb{F}_{q^n}$, a degree n extension. Then for a polynomial $\mathcal{F} \in \mathbb{F}_{q^n}[X]$*

$$\mathcal{F}(X) = \sum_{0 \leq i,j \leq n-1} \alpha_{i,j} X^{q^i+q^j} + \sum_{i=0}^{n-1} \beta_i X^{q^i} + \gamma_0,$$

with $\alpha_{i,j}.\beta_i, \gamma_0 \in \mathbb{F}_{q^n}$, there exists a unique system of quadratic polynomials $F = (f_1, \ldots, f_n) \in \mathbb{F}_q^n[x_1, \ldots, x_n]$, such that $\mathcal{F}(X) = \phi^{-1}(F(\phi(X)))$, where $\phi : \mathbb{F}_{q^n} \mapsto \mathbb{F}_q^n$ is a isomorphism as defined in Definition 3.1.1.

The *Oil and Vinegar* construction is another family of MQ schemes that was proposed by Patarin [Pat97]. This construction of the central trapdoor uses two sets of variables, called *oil* and *vinegar*. Let $o \in \mathbb{N}$ be the number of oil variables while $v \in \mathbb{N}$ be the number of vinegar variables, such that $n = (o + v)$. We represent the oil variables as $\{x_1, \ldots, x_o\}$ and the vinegar variables are represented as $\{x'_1, \ldots, x'_v\}$. We say that the polynomial $f(x_1, \ldots, x_o, x'_1, \ldots, x'_v) : \mathbb{F}_q^o \times \mathbb{F}_q^v \to \mathbb{F}_q$ below has a Oil and Vinegar shape if it has the following structure

$$f(x_1, \ldots, x_o, x'_1, \ldots, x'_v) = \sum_{0 \le i,j \le v} \alpha_{ij} x'_i x'_j + \sum_{0 \le i \le v, 0 \le j \le o} \alpha'_{ij} x'_i x_j + \sum_{0 \le i \le v} \beta_i x'_i + \sum_{0 \le i \le o} \beta'_i x_i + \gamma_0. \tag{3.1}$$

where $\alpha_{ij}, \alpha'_{ij}, \beta_i, \beta'_i, \gamma_0 \in \mathbb{F}_q$. The vinegar variables $(x'_1, \ldots, x'_v)$ can combine quadratically while the oil variables $(x_1, \ldots, x_o)$ do not mix with the oil variables.

To construct the cryptographic trapdoor based on this structure, one constructs a map $F : \mathbb{F}_q^{o+v} \to \mathbb{F}_q^o$,

$$F(x_1, \ldots, x_o, x'_1, \ldots, x'_v) = (f_1(x_1, \ldots, x_o, x'_1, \ldots, x'_v), \ldots, f_o(x_1, \ldots, x_o, x'_1, \ldots, x'_v)),$$

where the polynomials $(f_1, \ldots, f_o) \in \mathbb{F}_q^o[x_1, \ldots, x_o, x'_1, \ldots, x'_v]$ have the structure of Equation (3.1). This scheme is mostly useful in the design of digital signatures. Given a message $a \in \mathbb{F}_q^o$, one sets the vinegar variables to some random value $(b_1, \ldots, b_v) \in \mathbb{F}_q^v$ and solve the resulting system of linear equations

$$F(x_1, \ldots, x_o, b_1, \ldots, b_v) = a.$$

A non-zero solution to this system of equation is a candidate signature for the message a. A variant of this scheme was suggested with an unequal number of oil and vinegar variables ($o \neq v$), most famously known as the *Unbalanced Oil and Vinegar signature* (UOV) scheme [KPG99] after Kipnis and Shamir broke the balanced Oil and Vinegar scheme [KS98], which fixed $o = v = n/2$. This attack by Kipnis and Shamir also works on UOV in a probabilistic manner and has a complexity of $\mathcal{O}(q^{n-2o-1}o^4)$. So this scheme should have $v \geq 3o$ for a secure construction. The latest security evaluations for UOV can be found in [BWP05]. The Rainbow signature scheme [DS05] is another derivative of the UOV construction with multiple layers of oil and vinegar variables. Lifted-UOV (LUOV) [BPSV19] is a multivariate signature scheme that slightly modifies on the existing structure of UOV to reduce the size of public-keys. The scheme accomplishes this by lifting the public-key to an extension field. More specifically, instead of working over $\mathbb{F}_2$, LUOV works over an extension field $\mathbb{F}_{2^r}$ keeping the coefficients unchanged. Thus even though the public-key have the same coefficients, solving the system for some ciphertexts in $\mathbb{F}_{2^r}^m$ is more difficult than that for some ciphertext over $\mathbb{F}_2^m$. However, very recently an attack against LOUV was proposed [DZD+19]. The attack takes use of the fact

that the coefficients of the quadratic terms of the public-key polynomials are contained in a subfield $\mathbb{F}_{2^d}$ of $\mathbb{F}_{2^r}$. Taking use of a polynomial map from $\mathbb{F}_{2^d}^n$ to $\mathbb{F}_{2^r}^o$, the public-key can be transformed into another equivalent map over the subfield $\mathbb{F}_{2^d}$ over which it is easier to work. Therefore, the attack reduces forging a signature to solving an under-determined system of equations over the sub-field over which it is easier to find a solution. The attack broke the proposed parameters of `LOUV` and therefore required reparameterization.

Building on `HFE` and its variants, recently some new multivariate signatures were proposed in the `NIST` competition, that gathered some attention. Some multivariate signatures which have progressed through to the second round of the competition include `GeMSS` [CFMR+17], `LUOV`, and `Rainbow`, which we described before. Amongst the encryption candidates, there were only two multivariate based submissions: Giophantus [AGO+17] and `CFPKM` [CFP17], which was submitted by us. In Chapter 6, we describe `CFPKM` in great detail.

3.1.3 Generic Modifications on `MQ`-schemes

In the previous sections, we presented a variety of trapdoors that have served as foundation for most of the multivariate cryptographic schemes that have been proposed in the recent past. However, most of these trapdoors suffer from some vulnerabilities. Fortunately, many modifications are available, which can be used along with these. The general goal of these modifiers is to protect the design of such trapdoors from some commonly known attacks against multivariate schemes, which we shall discuss in the following sections.

3.1.3.1 Minus Modifier: "−"

This modifier, which was first witnessed in [Sha93], removes polynomials from the public-keys. In general one fixes $m' = m - a$ where $a \in \mathbb{N}$. The public-key is defined as the map $P := \pi \circ T \circ F \circ S$ where $\pi : \mathbb{F}_q^m \to \mathbb{F}_q^{m'}$ denotes a projection function, $S \in \mathrm{Aff}_n(\mathbb{F}_q), T \in \mathrm{Aff}_m(\mathbb{F}_q)$. The function π disregards the last a components of the output vector $(T \circ F \circ S) \in \mathbb{F}_q^m$.

The idea of introducing this modifier was a countermeasure against Patarin's linearization attack [PGC98] on $\mathtt{C}^*$. We discuss the linearization attack in Section 3.5. $\mathtt{C}^*$ with the minus modifier, written as $\mathtt{C}^{*-}$, was introduced by the name of `Flash` [PCG01a] in 2001. `Flash` along with a variant, named `Sflash` [PCG01a], were submitted to the European Nessie competition in 2001. The minus modifier was also seen as an answer to render the attacks [SK99, FJ03] against `HFE` useless.

It is important to note that, for a multivariate encryption scheme, on one hand, application of minus modifier increases the security of the scheme, on the other hand, it makes the decryption process inefficient. Removing a polynomials requires guessing q^a possible missing ciphertexts, where q is the size of the finite field over

which the public-keys are defined. Once we have all the possible solutions, one requires pruning through this solution set. Hence for efficient decryption, a must be small.

For digital signatures, a can be much larger. However, one must have enough public-keys in order that the underlying $\texttt{PoSSo}_q$ for this instance is still hard. In general, the choice of $a \leq n/2$ is efficient in practice.

Unfortunately, even with the application of this modifier, $\texttt{C}^{*-}$ and its variants have broken in [DFSS07, BFMR11, GM02]. Application of this modifier on HFE ($\texttt{HFE}^-$) [PGC98] with some appropriate parameters renders the attacks of [SK99, FJ03] ineffective.

3.1.3.2 Plus Modifier: "+"

By the suggestion of the name, this modifier adds polynomials to the public-keys. This method was first observed in [PGC98]. A legitimate user adds $a \in \mathbb{N}$ random quadratic equations to the public-keys. For the construction, one sets $m' = m + a$ and then defines the public-key as $P := T \circ (F^+ \mid\mid F) \circ S$ where $F^+ \in \mathbb{F}_q^a[x]$ and $T \in \mathbb{F}_q^{m' \times m'}$. This idea was used as a technique to thwart differential attacks in [DG06].

For a multivariate signature scheme, a must be small. This is because, given a signature, the probability of satisfying the additional a public-keys is $1/q^a$. For small q and a, after at most q^a tries, we obtain the signature.

For an encryption scheme, a can be larger. However, the number of public-keys $m + a$ must be such that the problem of solving the system of equations is intractable. That is the number of equations must be less than $\frac{n(n+1)}{2}$ since Gaussian elimination solves this system by re-linearization. In general, $a < n/2$ is efficient in practice. Additionally, this modifier is also used in conjunction with the minus modifier to make cryptosystems behave close to generic systems [PGC98].

3.1.3.3 Projection Modifier: "p"

The idea of this modifier is to fix a certain fraction of variables in the public-keys. The projection modifier projects the n-dimensional input vector onto a $(n-k)$ dimensional space before passing through the central map, i.e., we take $(n-k)$ blocks of plaintext and then evaluate the public-key values over

$$P = (p_1(x_1, \ldots, x_{n-k}, 0, \ldots, 0), \ldots, p_m(x_1, \ldots, x_{n-k}, 0, \ldots, 0)),$$

where $(p_1, \ldots, p_m) \in \mathbb{F}_q^m[x_1, \ldots, x_n]$. Dubios showed that just adding minus modifier was not enough to make the $\texttt{C}^*$ secure as there exists some kind of differential symmetry [DFSS07]. Therefore, this idea of projecting the public-keys to a lower dimensional subspace was proposed [DYC+07]. This modifier is used in conjunction with other modifiers, e.g., the minus modifier.

3.1.3.4 Vinegar variables: "v"

This perturbation was introduced for HFE in [KPG99]. The main idea of this perturbation was to hide the structure of the HFE map by introducing additional variables, known as vinegar variables. The way the modification works is as follows: One chooses $v \in \mathbb{N}$ vinegar variables, $(x'_1, \ldots, x'_v)$ in addition to the already existing variables $(x_1, \ldots, x_n)$. A HFEv polynomial is defined over $\mathbb{F}_q[x_1, \ldots, x_n, x'_1, \ldots, x'_v]$ and has the following structure

$$P(x_1, \ldots, x_n, x'_1, \ldots, x'_v) = \sum_{0 \leq i,j < n} \alpha_{ij} x_i x_j + \sum_{0 \leq i < n} \sum_{0 \leq j < v} \beta_{ij} x_i x'_j + \sum_{0 \leq i,j < v} \gamma_{ij} x'_i x'_j,$$

where $\alpha_{ij}, \beta_{ij}, \gamma_{ij} \in \mathbb{F}_q$. Consider a cryptosystem which applies the vinegar modifier on its central trapdoor polynomials. Inverting the new central trapdoor equation for a fixed $y \in \mathbb{F}_q^n$, requires inversion of the original trapdoor for q^v different values of the vinegar variables $(x'_1, \ldots, x'_v)$. For a multivariate signature scheme, this is not an issue as finding any solution will produce a valid signature. Unfortunately, in the case of encryption, this design is not suitable.

This modifier served as a foundation for the previously described oil and vinegar construction. The Vinegar perturbation has also been used in conjunction with the minus modifier over HFE (denoted as HFEv^-) and has been used extensively for constructing multivariate signature schemes, e.g., Quartz [PCG01b], GUI [PCY+15, PCY+17] and GeMSS [CFMR+17].

Table 3.1 gives an assortment of some of the famous as well as newly proposed multivariate encryption and signature primitives whose constructions are based on the above discussed modifiers applied to the basic multivariate constructions. In the following section, we shall discuss the design of a multivariate encryption scheme known as Extension Field Cancellation [SDP16]. One of the major contributions of this thesis is the detailed cryptanalysis of this scheme, the details of which can be found in Chapter 4.

3.1.4 EFC Scheme

The Extension Field Cancellation scheme [SDP16] is a MQ based encryption scheme which was proposed by Alan Szepieniec *et al.* at PQCrypto 2016. EFC scheme mixes the arithmetic over $\mathbb{F}_q$ along with arithmetic over degree n extension field $\mathbb{F}_{q^n}$. Let $\varphi : \mathbb{F}_q^n \to \mathbb{F}_{q^n}$ be a $\mathbb{F}_q$-vector space isomorphism. Let $A, B \in \mathbb{F}_q^{n \times n}$ and denote $\mathbf{x} = (x_1, \ldots, x_n)$ as a vector of variables over $\mathbb{F}_q[\mathbf{x}]$. We can represent the multivariate polynomial ring $\mathbb{F}_{q^n}[\mathbf{x}]$ as a univariate polynomial ring $\mathbb{F}_{q^n}[\chi]$ where, using φ one can create the extension field variable $\chi = \varphi(\mathbf{x})$ from a vector of intermediates $\mathbf{x} \in \mathbb{F}_q^n$. Let $\alpha(\mathbf{x}) = \varphi(A\mathbf{x}) \in \mathbb{F}_{q^n}[\mathbf{x}]$ and $\beta(\mathbf{x}) = \varphi(B\mathbf{x}) \in \mathbb{F}_{q^n}[\mathbf{x}]$. The square matrix that represents multiplication by $\varphi(A\mathbf{x})$ is denoted by $\alpha_m(\mathbf{x}) \in$

Flash, Sflash	A signature scheme [PCG01a,PGC98] based on C^{*-} also submitted to Nessie. Was broken extensively in [DFSS07, BFMR11, GM02]
Pflash	A secure signature scheme for smart cards based on C^* with projection and minus modifications [CYST15, CST17].
Eflash	An encryption scheme [CST18] based on HFE with the idea of evading the standard attacks of MinRank and differential attacks. Was broken recently using Gröbner basis techniques [ØFRC20].
Quartz	A 128 bit signature scheme based on $HFEv^-$, submitted to the European Nessie call for cryptographic schemes [PCG01b]. [CDF03] discusses the security of the scheme against "inversion" attacks.
GeMSS, GUI	Signature schemes [CFMR+17] and [PCY+17], based on $HFEv^-$ recently proposed to the NIST standardization competition.
ZHFE	An encryption scheme [PBD14] that utilizes a combination of pair of high degree HFE style polynomials to find a third low degree polynomial. Decryption requires the inversion of this low degree polynomial. [CSTV] proposed a low rank equivalent key recovery attack, extracting a private key from the ZHFE public-key.
HFERP	An encryption scheme [IPST+18] that combines the HFE and Rainbow to compose the central trapdoor.
Simple Matrix, Cubic Simple Matrix	Encryption schemes [TDTD13] and [DPW14] introduce polynomial matrix arithmetic for construction of the pubic key polynomials. The decryption involves inverting (invertible with high probability) the polynomial matrices. [Gu16] proposed a key recovery attacks against [TDTD13]. Similar key recovery attack on [DPW14] was proposed by [MPST16].

Table 3.1 – List of multivariate cryptosystems.

$\mathbb{F}_q^{n\times n}$ (See Appendix C for an example to compute $\alpha_m(\mathbf{x})$) . The central map for EFC [SDP16] can be defined in two equivalent forms F and $\mathcal{F}$ as

$$F : \mathbb{F}_q^n \to \mathbb{F}_q^{2n} : \mathbf{x} \to (\alpha_m(\mathbf{x})\mathbf{x}\,,\, \beta_m(\mathbf{x})\mathbf{x}),$$

$$\mathcal{F} : \mathbb{F}_{q^n} \to \mathbb{F}_{q^n}^2 : \chi \to (\alpha(\chi)\chi\,,\, \beta(\chi)\chi). \tag{3.2}$$

where $F = (\varphi^{-1}, \varphi^{-1}) \circ \mathcal{F} \circ \varphi$. The public-key of EFC $P : \mathbb{F}_q^n \mapsto \mathbb{F}_q^{2n}$ is a composition of this central map F along with two external invertible affine transformations $S \in \mathrm{Aff}_n(\mathbb{F}_q)$ and $T \in \mathrm{Aff}_{2n}(\mathbb{F}_q)$, i.e., $P = T \circ F \circ S$.

Now let us assume $(c_1, \ldots, c_{2n}) = P(s_1, \ldots, s_n) \in \mathbb{F}_q^{2n}$ be the encryption of a secret message $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{F}_q^n$. Let $\mathbf{d_1'} = (c_1, \ldots, c_n) \in \mathbb{F}_q^n$ and $\mathbf{d_2'} = (c_{n+1}, \ldots, c_{2n}) \in \mathbb{F}_q^n$. So $P(\mathbf{s}) = (\mathbf{d_1'}\,,\, \mathbf{d_2'})$. Let $(\mathbf{d_1}\,,\, \mathbf{d_2}) = T^{-1}(\mathbf{d_1'}\,,\, \mathbf{d_2'})$. Decryption for EFC involves solving the system of linear equations

$$\beta_m(\mathbf{x})\mathbf{d_1} - \alpha_m(\mathbf{x})\mathbf{d_2} = \mathbf{0}. \tag{3.3}$$

The design of the above secret-key of EFC from the public-keys and ciphertexts ensures that Gaussian elimination on this system of linear equations will find at least one solution. Now this solution may not be unique, but according to [SDP16], they expect the set of solutions to be small.

3.1.4.1 Minus with projection modifier ($\mathtt{EFC}_q^-(a)$)

[SDP16] takes use of the "minus" modifier as discussed previously in Section 3.1.3.1. Also in anticipation of differential symmetry attacks [DFSS07, BFMR11], EFC use the "projection" variant (see Section 3.1.3.3) in conjunction with minus modifier. The $\mathtt{EFC}^-$ system with projection of one variable has public-keys as the system of $2n - a$ quadratic polynomials in $n - 1$ variables.

$$P^- = \big(p_1(x_1, \ldots, x_{n-1}, 0), \ldots, p_{2n-a}(x_1, \ldots, x_{n-1}, 0)\big).$$

We denote the minus modifier as $\mathtt{EFC}_q^-(a)$ where the EFC is defined over $\mathbb{F}_q$ and a equations are removed.

3.1.4.2 Frobenius Tail modifier ($\mathtt{EFC}_q^F$)

Another modifier proposed with the central map of EFC was the *Frobenius tail modifier*. This modifier adds an extra quadratic term $\beta^3(\chi) \in \mathbb{F}_{q^n}[\chi]$ and $\alpha^3(\chi) \in \mathbb{F}_{q^n}[\chi]$ to $\alpha(\chi)\chi$ and $\beta(\chi)\chi$ respectively.

$$\mathcal{F}' : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}^2 : \chi \to (\,\alpha(\chi)\chi + \beta^3(\chi)\,,\, \beta(\chi)\chi + \alpha^3(\chi)\,).$$

For the Frobenius modified EFC central map, the decryption is very similar to that of non-perturbed case of EFC. We saw that the public-keys over the extension field

$\mathbb{F}_{q^n}[\chi]$ are represented as $\alpha(\chi)\chi+\beta^3(\chi)$ and $\beta(\chi)\chi+\alpha^3(\chi)$ and let the corresponding ciphertext be represented as $\mathcal{D}_1, \mathcal{D}_2 \in \mathbb{F}_{q^n}$. The decryption for $\texttt{EFC}_q^F$ involves solving a system of linear equations over $\mathbb{F}_q^n$ that, over the extension field $\mathbb{F}_{q^n}$, is represented by the following univariate polynomial equation

$$\beta(\chi)\mathcal{D}_1 - \alpha(\chi)\mathcal{D}_2 = \beta^4(\chi) - \alpha^4(\chi). \tag{3.4}$$

The problem with the minus modifier is that the decryption complexity is exponential to the number of polynomials removed, hence higher the a, more time decryption takes while on the other hand decreasing a, reduces security of the scheme. The rationale suggested behind adding this modifier is to achieve the same level of security for an EFC instance but with a low value of a, allowing efficient decryption.

3.1.4.3 Proposed Challenge Parameters

To ensure at least 80 bits of security, [SDP16] proposed the following three parameters. The first and the third parameter sets have the projection modifier added to it, while the second parameter has the projection as well as the Frobenius tail modifier in the definition of the public-keys.

Challenge	q	n	a	Modifiers
1	2	83	10	Minus, Projection
2	2	83	8	Minus, Projection & Frobenius
3	3	59	6	Minus, Porjection

Table 3.2 – Challenge Parameters EFC [SDP16]

3.2 Standard attacks on MPKCs

Having looked at some of the state of the art multivariate primitives, in this section, we will have a brief look at some standard line of attacks that have been used by cryptographers in order to find vulnerabilities of these cryptosystems. Broadly, attacks on multivariate schemes can be classified into *Key Recovery attacks* and *Message Recovery attacks*. In the following sections, we shall give a brief overview of some of the most prominent key and message recovery attacks that are generally the primary form of attacks on multivariate schemes.

3.2.1 Key Recovery Attacks

The first class of attacks focuses on recovering the secret-keys of a multivariate scheme. Recall from Section 3.1.1, the public-key is a composition of three transformations: two invertible linear transformations $S \in \mathbb{F}_q^{n\times n}, T \in \mathbb{F}_q^{m\times m}$ and a

quadratic map $F : \mathbb{F}_q^n \mapsto \mathbb{F}_q^m$. Key recovery attacks focus on retrieving these maps by exploiting the hidden structural vulnerabilities in the design of the public-keys of the multivariate schemes. In the following subsection, we shall describe three major types of key-recovery attacks.

3.2.1.1 Linearization attack

We recall that the central map equation of the Matsumoto-Imai scheme C* is given by $y = x^{q^\alpha+1}$, for some $\alpha \in \mathbb{N}$. However, an algebraic implication of this equation is that

$$y^{q^\alpha-1} = x^{q^{2\alpha}-1} \implies xy^{q^\alpha} = yx^{q^{2\alpha}}. \tag{3.5}$$

Hence over $\mathbb{F}_q$, these are equations of the form

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\beta_{ij}x_iy_j + \sum_{i=1}^{n}\gamma_ix_i + \sum_{i=1}^{n}\delta_iy_i + \zeta = 0, \tag{3.6}$$

where $\beta_{ij}, \gamma_i, \delta_i, \zeta \in \mathbb{F}_q$, $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ are the equivalent base field vector representations of x and y respectively (all x_i and y_i are elements of $\mathbb{F}_q$). These equations hold for all pairs of plaintexts x and ciphertexts y. Hence given enough plaintexts-ciphertexts pairs (x, y), we obtain linear equations in $(n+1)^2$ variables $\beta_{ij}, \gamma_i, \delta_i$ and ζ, which are the coefficients of equation (3.6). These coefficients can be obtained by Gaussian elimination over these linear equations. Once the equations are recovered, we can substitute the value of ciphertexts into $(y_1, \ldots, y_n)$ yielding a linear system of equations in variables $(x_1, \ldots, x_n)$. Gaussian elimination allows us to recover the kernel whose size depends on the number of independent equations. This attack was proposed by Patarin in [Pat95].

3.2.1.2 Differential Attacks

For a MQ central map $F \in \mathbb{F}_{q^n}[x]$, consider the differential $dF_k(x) = F(x+k) - F(x)$, where $k \in \mathbb{F}_{q^n}$. Now, let us consider the operation $DF = dF_k(x) - dF_k(0)$. Since F is quadratic over the base field $\mathbb{F}_q^n$, hence the differential is linear and so is the operation DF. We can also write

$$DF = F(x+k) - F(x) - F(k) + F(0).$$

This differential proves handy for cryptanalysis of many multivariate cryptosystems. Even so, the previously mentioned Patarin's linearization attack can be seen as a form of a differential attack. For $f(x) = x^{q^\alpha}$, the differential Df is a symmetric bilinear function

$$Df(y, x^{q^\alpha+1}) = yx^{q^{2\alpha}+q^\alpha} + y^{q^\alpha}x^{q^\alpha+1} = x^{q^\alpha}\left(yx^{q^{2\alpha}} + xy^{q^\alpha}\right) = 0.$$

Dividing both sides by x^{q^α}, we get Equation (3.5). Fouque *et al.* [FGS05] show that C* and C*− exhibit multiplicative symmetry in their bilinear differential which when further analyzed on their rank or kernel gave insights of the secret-key. This attack also works on the Perturbed C* [Din04] which was proposed by Jintai Ding.

Another example of the differential attack is the differential cryptanalysis of the Hidden Matrix (HM) cryptosystem by Faugère *et al.* [FJPT10]. The central map of HM is given by

$$F(X) = X^2 + M \cdot X, \quad \text{with } X, M \in \mathcal{M}_n(\mathbb{F}_q),$$

where $\mathcal{M}_n(\mathbb{F}_q)$ is the set of matrices of size $n \times n$ over $\mathbb{F}_q$. Applying the differential gives us $DF(X,Y) = X \cdot Y + Y \cdot X$. Fixing $X = X_0 \in \mathcal{M}_n(\mathbb{F}_q)$ we get a system of n^2 linear equations in n^2 coefficients of Y. Depending on the number of solutions to this system, they show that the system of polynomials in the public-key of HM has a behavior that is totally unlike a random system of equations. We refer the reader to [FJPT10] for more details.

3.2.1.3 Rank attacks

In order to understand rank based attacks on such central maps, one quantity which is most vital is the quadratic rank of the public-key.

Definition 3.2.1 (Quadratic rank). *The quadratic rank (or Q-rank) of a multivariate quadratic map $f : \mathbb{F}_q^n \to \mathbb{F}_q^n$ is the rank of the quadratic form $\mathcal{Q}$ on $\mathbb{F}_{q^n}[X]$, defined by $\mathcal{Q} = \phi \circ f \circ \phi^{-1}(X)$, under the identification $X = \phi^{-1}(x)$.*

This term is important from a cryptanalytic point of view is because Q-rank is not invariant under a linear transformation. However, the minimum Q-rank, which is minimum possible rank observed amongst all linear images of the quadratic map f, remains invariant. Consider the external linear transformations S and T lifted from $\mathbb{F}_q^n$ to $\mathbb{F}_{q^n}$ and thus have the form

$$S(X) = \sum_{i=0}^{n-1} s_i X^{q^i}, \; T^{-1}(X) = \sum_{i=0}^{n-1} t_i X^{q^i},$$

where $s_i, t_i \in \mathbb{F}_{q^n}$. This allows representing the public-key P of the HFE cryptosystem over the extension field $\mathbb{F}_{q^n}$, $P(X) = T(F(S(X)))$ with a univariate representation. Now consider the public-key polynomial $P(X)$ in the matrix form written as follows:

$$P(X) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} p_{ij} X^{q^i+q^j} = \underline{X} \mathtt{P} \underline{X}^t,$$

where $\mathtt{P} = [p_{ij}]$ is a $n \times n$ matrix over $\mathbb{F}_{q^n}$, $\underline{X} = (X, X^q, \ldots, X^{q^{n-1}})$ is a vector over $\mathbb{F}_{q^n}$ and $\underline{X}^t$ is its transpose. This implies that

$$T^{-1}(P(X)) = \sum_{k=0}^{n-1} t_k \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (p_{i-k,j-k})^{q^k} X^{q^i+q^j},$$

$$F(S(X)) = \underline{X}\mathtt{WFW}^t\underline{X}^t,$$

where $\mathtt{F}$ denotes the $\mathtt{HFE}$ central map in matrix form and $\mathtt{W} = [s_{j-i}^{q^i}]$ is another matrix. Let $\mathtt{P}^{*k}$ be the matrix obtained from $\mathtt{P}$ by raising all entries of $\mathtt{P}$ to their q^k power and cyclically rotating all rows and columns of $\mathtt{P}$ forward by k steps. Thus we obtain

$$T^{-1}(P(X)) = \underline{X}\mathtt{P}'\underline{X}^t$$

where,

$$\mathtt{P}' = \sum_{k=0}^{n-1} t_k \mathtt{P}^{*k} = \mathtt{WFW}^t. \tag{3.7}$$

Considering $\mathtt{HFE}$, the homogeneous quadratic part of central map F can be written as

$$\begin{bmatrix} X & X^q & \cdots & X^{q^{n-1}} \end{bmatrix} \begin{bmatrix} \alpha_{0,0} & \alpha'_{0,1} & \cdots & \alpha'_{0,D-1} & 0 & \cdots & 0 \\ \alpha'_{0,1} & \alpha_{1,1} & \cdots & \alpha'_{1,D-1} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \alpha'_{0,D-1} & \alpha_{1,1} & \cdots & \alpha_{D-1,D-1} & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} X \\ X^q \\ \vdots \\ X^{q^{n-1}} \end{bmatrix}$$

where $\alpha'_{i,j} = \frac{1}{2}\alpha_{i,j}$ and $D = \lceil \log_q d \rceil$. The rank of P is bounded by $r = D + 1$, and thus the quadratic rank of the $\mathtt{HFE}$ polynomial. As said this quadratic rank is invariant under isomorphism of polynomials, hence the rank of $\mathtt{WFW}^t$ is also bounded by r. [SK99] showed that given the correct choices of the values of $t_o, \ldots, t_{n-1}$, the rank of $\mathtt{P}'$ is bounded by r. Since the rank is not more than r, therefore the left kernel of $\mathtt{P}'$ is at least a $n-r$ dimensional vector subspace. Assigning random values to the $n-r$ entries and creating new variables for the rest of r variables, one obtains $n(n-r)$ quadratic equations over $n + (n-r)r$ variables. [SK99] proposed to solve this by re-linearization techniques that was later improved by another linearization technique, XL [CKPS00]. Later Nicolas Courtois [Cou01] showed that there is an equivalence between this attack and that of a MinRank kernel attack. Additionally, he also proposed converting the problem of recovering the linear transformation T into the solution of a MinRank problem (which is $\mathtt{NP}$-Complete) over $\mathbb{F}_{q^n}$. Taking $(t_0, \ldots, t_{n-1})$ as variables, one considers all the $(r+1) \times (r+1)$ minors of $\mathtt{P}'$ that have determinant 0. One can solve these system of $\binom{n}{r+1}^2$ equations with $\binom{n}{r+1}$ monomials and this can be solved by Gaussian elimination.

An improvement to this previous attack was proposed by Luk Bettale *et al.* [BFP13]. Instead of solving the equations from $(r+1)$-minors by linearization and Gaussian elimination, [BFP13] proposed using Gröbner basis techniques to solve these equations over the variables $(t_0, \ldots, t_{n-1})$. [BFP13] also differs from the original method of using the univariate polynomial representation of the public-keys

by using the multivariate representation of the public-keys with the coefficients represented over the base field. This is known as the *minors modeling* of the Kipnis-Shamir attack. The asymptotic complexity of this attack is $\mathcal{O}(n^{(D+1)\omega})$ where ω is the linear algebra constant.

This approach also works with the minus variants of HFE. Removing one equation from the public-key leads to an increase in the quadratic rank by one. Until the number of removed equation is small enough compared to n, the minors modeling attack still holds, but the asymptotic complexity increases to $\mathcal{O}(n^{(\log_q d+1+a)\omega})$ where a is the number of public equations removed.

3.2.2 Message Recovery Attacks

From Chapter 2, we recall the discussions of the various methods to solve $\mathtt{PoSSo}_q$. Naturally, such techniques are useful in mounting attacks to recover the hidden secret from the multivariate schemes, since from Section 3.1 we know such cryptosystems are based on $\mathtt{PoSSo}_q$.

3.2.2.1 Exhaustive Search

In Section 2.2.1, we presented the state of the art combinatorial algorithms for the $\mathtt{PoSSo}_q$ problem. The total number of possible values that the variables in public-key equations can have is q^n. For each possible value, the corresponding ciphertext is evaluated and checked against the input ciphertext. The candidate solution with matching ciphertext is the correct solution. However, such attacks are usually exponential in the number of variables. As seen earlier in Section 2.3.1, this technique can be used in combination with other message recovery attacks, such as Gröbner basis attacks.

3.2.2.2 Gröbner Basis Algorithms

Recalling from Chapter 2, Gröbner basis computation also provides us with a method to compute the solution to a system of polynomial equations. Hence multivariate cryptographic primitives are vulnerable to direct algebraic attacks by computing the Gröbner basis.

One common observation from the previously discussed multivariate cryptosystems is that the public-keys are far more structured than a generic or random system of polynomials. We know that the degree of regularity for a regular system is given by the index of the first non-positive coefficient of the Hilbert series (see property 2.3.31). For a structured semi-regular system, the above defined degree of regularity works as an upper bound for the maximal degree observed in the Gröbner basis computation. However, more accurate computations of this degree of regularity for multivariate schemes have made some advances in recent years.

A prime example is the new upper bound for the degree of regularity for HFE and HFE$^-$ proposed by Ding *et al.* [DK12].

Theorem 3.2.2. *[DK12] Let $\mathcal{Q}$ be the quadratic map of HFE. If the Q-rank$(\mathcal{Q}) > 1$, the degree of regularity of the system is upper bounded by*

$$\frac{(q-1)(\lfloor \log_q(d-1) \rfloor + 1)}{2} + 2.$$

If Q-rank$(\mathcal{Q}) = 1$, then the degree of regularity is less than equal to q.
Let $r = \lfloor \log_q(d-1) \rfloor + 1$. The degree of regularity for HFE$^-$ with a equations removed is upper bounded by

$$\frac{(q-1)(\lfloor \log_q(d-1) \rfloor + 1 + a)}{2} + 2,$$

if q is odd or if q is even and $r + a$ is even,

$$\frac{(q-1)(\lfloor \log_q(d-1) \rfloor + a)}{2} + 2,$$

if q is even and $r + a$ is odd.

Although these upper bounds do give an idea about the degree of regularity for a system of public-keys, however, these are not always tight, as we have encountered in the cryptanalysis of EFC (see Chapter 3.1.4). Now, let us see why is this important for any multivariate cryptosystem. When any such multivariate scheme is designed, the complexity of algebraic attacks plays the most important role in setting the secure parameters for the scheme. This is because, in event of any such algebraic attacks being launched, the hardness of recovering the secret message without the use of the private key remains intact. We shall also recall that the complexity of the algebraic attack on such systems is directly related to the degree of regularity of the public-keys.

Unfortunately, most of the current multivariate schemes use non-tight upper bounds, such as the one in Theorem 3.2.2, for the degree of regularities. Thus from a cryptanalytic point of view, it becomes quite imperative to get the exact degree of regularities of the multivariate schemes. Accurate measurement of the degree of regularity leads us to determine the complexity of algebraic attacks with good accuracy. Hence experimental analysis is a modus operandi for this.

As discussed in Chapter 2, MAGMA proves quite useful to gather the degree of regularity from standard Gröbner basis algorithms. Especially, the F4 algorithm on MAGMA provides an incremental degree by degree computation of bases until no new relations are recovered. In Figures 3.1 and 3.2, we provide a snippet of a Gröbner basis computation on MAGMA. Here there are certain observations, firstly for each degree, the F4 algorithm enumerates the number of new relations observed,

indicating the total degrees of such new relations. Secondly, the Gröbner basis computation terminates after it has recovered linearly independent linear equations that belong to the same ideal of the public-keys. The degree of regularity is the highest degree that is reached during this process before we recover close to full rank set of linear equations. This is because it is the maximal degree after which there are no new polynomials that are discovered and thus no new basis elements add to the already recovered Gröbner basis elements for any degree greater than this degree.

For the case of an non-homogeneous or affine system of polynomials, there is an apparition of lower degree relations at some particular step degree. For example, looking at Figure 3.1, in line 46, we observe 43 new quadratic polynomials in step degree 3. Similarly in Figure 3.2, line 31 shows that in degree 4, there is apparition of 41 linear polynomials. In this thesis, we use these experimental observations to perform cryptanalysis of a multivariate scheme. Thus, from this section, we have a general idea about multivariate cryptography: state of the art design of schemes, new ways of designing schemes from the pre-existing schemes using modifiers and the attacks on multivariate primitives.

In this thesis, we have present a new multivariate key encapsulation scheme, namely `CFPKM`. It has been presented in great detail in Chapter 6. The design of the scheme is loosely based on a lattice-based key exchange primitive called `Frodo`. Therefore, for understanding the working of the information exchange, it is important to have a look at this scheme. This scheme is based on a lattice based problem, the Learning With Errors problem [Reg09]. In this following section we shall very briefly present the `LWE` problem and the `Frodo` key-exchange scheme.

3.3 Lattice Based Cryptosystems

Among all the post-quantum computational problems, lattice-based problems have received a bulk of the attention from researchers in the past decade. Lattice problems have the advantage of worst-case to average-case reduction. This informally means that private keys in the easiest case are as hard to break as in the worst case. For example, take an instance of RSA, where the choice of the keys involves choosing two large random primes and expect that this yields a hard instance of integer factorization problem. However, there is a probability of choosing the wrong pair, resulting in a lower level of security. In lattice based cryptosystem, all possible key choices are equally hard to solve.

There are several NP hard lattice problems, namely, Shortest (Approximate) Vector problem (`SVP` and $\texttt{SVP}_\gamma$) [Ajt96], Bounded Distance Decoding (`BDD`) [LM09] and Closest Vector Problem (`CVP`) [GMSS99] whose hardness assumptions acts as a security for lattice based cryptographic primitives against classical as well as quantum adversaries. One of the most recent lattice problem is Learning With

```
********************
FAUGERE F4 ALGORITHM
********************
Coefficient ring: GF(2)
Rank: 42
Order: Graded Reverse Lexicographical (bit vector)
Reduced exponents (solution over GF(2))
Matrix kind: Packed GF(2)
Datum size: 0
No queue sort
Stop at 10 linear(s)
Initial length: 239
Inhomogeneous
Initial queue setup time: 0.009
Initial queue length: 472
*******
STEP 1
Basis length: 156, queue length: 472, step degree: 2, num pairs:
   149
Basis total mons: 67006, average length: 429.526
0 field pair(s)
Number of pair polynomials: 149, at 862 column(s), 0.000
Average length for reductees: 429.68 [149], reductors: 426.14 [7]
Symbolic reduction time: 0.000, column sort time: 0.000
149 + 7 = 156 rows / 862 columns out of 904 (95.354%)
Density: 49.829% / 49.883% (429.53/r), total: 67006 (0.3MB)
Before ech memory: 32.1MB (=max)
Row sort time: 0.000
0.000 + 0.000 + 0.000 = 0.000 [149]
After ech memory: 32.1MB (=max)
Num new polynomials: 149, min deg: 2 [149], av deg: 2.0
*******
STEP 2
Basis length: 305, queue length: 2293, step degree: 3, num pairs:
   2293
Basis total mons: 119696, average length: 392.446
298 pairs eliminated
312 field pair(s)
Number of pair polynomials: 1999, at 9982 column(s), 0.079
Average length for reductees: 406.40 [1999], reductors: 358.23
   [4557]
Symbolic reduction time: 0.089, column sort time: 0.010
1999 + 4557 = 6556 rows / 11522 columns out of 12384 (93.039%)
Density: 3.2366% / 6.1727% (372.92/r), total: 2444868 (9.3MB)
Before ech memory: 64.1MB (=max)
Row sort time: 0.000
0.269 + 0.000 + 0.260 = 0.530 [1995]
After ech memory: 64.1MB (=max)
Num new polynomials: 1995, min deg: 2 [43], av deg: 3.0
```

Figure 3.1 – A sample of Gröbner basis computation process on MAGMA : part 1

```
*******
STEP 3
Basis length: 2300, queue length: 43383, step degree: 3, num pairs
    : 1428
Basis total mons: 4941802, average length: 2148.610
86 field pair(s)
Number of pair polynomials: 1428, at 10703 column(s), 0.120
Average length for reductees: 1187.00 [1428], reductors: 837.16
    [6068]
Symbolic reduction time: 0.289, column sort time: 0.000
1428 + 6068 = 7496 rows / 10703 columns out of 12384 (86.426%)
Density: 8.4444% / 13.583% (903.81/r), total: 6774940 (25.8MB)
Before ech memory: 160.2MB (=max)
Row sort time: 0.000
0.219 + 0.000 + 0.100 = 0.330 [1428]
After ech memory: 160.2MB (=max)
Num new polynomials: 1428, min deg: 3 [1428], av deg: 3.0
Queue insertion time: 2.309
New max step: 3, time: 3.050
Step 3 time: 3.050, [3.053], mat/total: 0.860/5.059, mem: 160.2MB
    (=max)

*******
STEP 4
Basis length: 3728, queue length: 100252, step degree: 4, num
    pairs: 90708
Basis total mons: 7216638, average length: 1935.793
10140 field pair(s)
Number of pair polynomials: 90708, at 84997 column(s), 24.609
Average length for reductees: 2203.94 [90708], reductors: 1028.65
    [76975]
Symbolic reduction time: 2.990, column sort time: 0.079
90708 + 76975 = 167683 rows / 84997 columns out of 124314
    (68.373%)
Density: 1.9582% / 3.6994% (1664.4/r), total: 279095438 (1064.7MB)
Before ech memory: 1313.3MB (=max)
Row sort time: 0.039
Found 41 linear(s) having done 8192 of 90708
Linear found
6.100 + 0.010 + 0.000 = 6.140 [41]
Number of unused reductors: 35807
After ech memory: 1392.8MB (=max)
Num new polynomials: 41, min deg: 1 [41], av deg: 1.0
Queue insertion time: 0.030
Number of linears: 41
New max step: 4, time: 33.900
Step 4 time: 33.900, [33.915], mat/total: 7.040/38.960, mem:
    1377.4MB, max: 1392.8MB
STOP at 41 linears
```

Figure 3.2 – A sample of Gröbner basis computation process on MAGMA : part 2

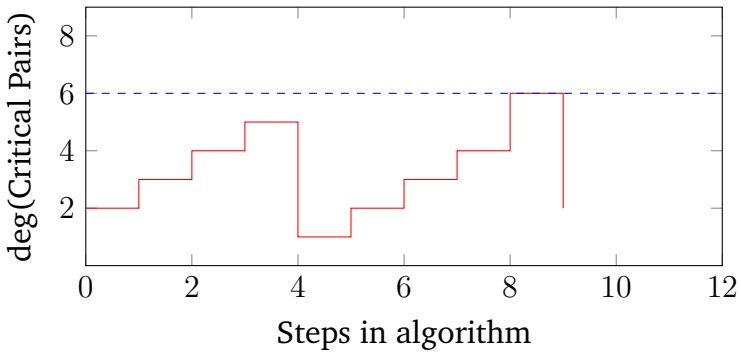

Figure 3.3 – A general behaviour of degree of critical pairs in affine case

Error (LWE) which was proposed by Regev in 2005 [Reg09]. Let χ be a probability distribution over a set S, then $x \leftarrow \chi$ denotes a sampling of $x \in S$ according to χ. The LWE problem is defined as follows:

Definition 3.3.1 (LWE problem). *[Reg09] For positive integers n and $q \geq 2$, a vector $s \in \mathbb{Z}_q^n$, and a probability distribution χ on $\mathbb{Z}_q$, we define $A_{s,\chi}$ to be the distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by choosing a vector $a \in \mathbb{Z}_q^n$ uniformly at random, an error term, $e \leftarrow \chi$, and that outputs $(a, \langle a, s \rangle + e)$.*
For a dimension n, an integer q and an error distribution χ over $\mathbb{Z}_q$, the learning with errors problem $\mathtt{LWE}_{n,q,\chi}$ is defined as follows: given samples from $A_{s,\chi}$ for some arbitrary $s \in \mathbb{Z}_q^n$, recover s with probability exponentially close to 1.

This problem has been quite extensively studied in the past few years in [ACPS09, LP11, BLP+13] to name a few. The hardness of the LWE problem is known to be based on the worst-case hardness of standard lattice problems such as Decision Shortest Vector Problem (GapSVP) [Reg09, Pei09]. Particularly, Piekert showed that when the modulus q is exponential, LWE has a classical reduction from GapSVP [Pei09]. Thus the hardness is based on the standard assumption that GapSVP is hard to approximate to within polynomial factors. For the case when the modulus q is polynomial in n, [BLP+13] gave a classical reduction from the GapSVP in dimension $\sqrt{n}$ to an LWE instance in dimension n.

Lattices have been heavily used in designing both encryption/decryption as well as signature schemes in the recent past. Based on LWE and its variants many encryption/decryption schemes have been proposed, for example NewHope [AAB+]. Recently, numerous proposals based on lattice based problems and especially LWE were submitted to the NIST competition: Crystal-Kyber-Dilithium [ABD+17], Round5 [BGML+18], Saber [DKRV18], NewHope [AAB+] to name a few. In this following subsection, we shall describe Frodo [BCD+16]. This scheme was also submitted as a Key-encapsulation/ Encryption scheme (FrodoKEM [ABD+18]) in the NIST competition, however, we must mention that FrodoKEM differs slightly from the version adapted in the following subsection. For more information on the

FrodoKEM scheme, we redirect the reader to [ABD+18].

3.3.1 Frodo Key Exchange

3.3.1.1 Notation

Matrices are denoted by bold capital letters, e.g $\mathbf{A}, \mathbf{B}$. For a distribution χ, $\mathbf{A} \leftarrow \chi(S^{n \times m})$ defines a matrix with each component chosen independently according to the distribution $\chi(S)$. As seen previously, the LWE problem is characterized by three parameters: the modulus q, the dimensions of the matrix n, and an error distribution χ. For designing this scheme, the authors of [BCD+16] choose a discrete Gaussian distribution that is defined as follows.

Definition 3.3.2 (Discrete Gaussian Distribution)**.** *Let α be a real number and $q \in \mathbb{N}$. A Gaussian distribution is a continuous probability distribution function that centers around a mean value with a standard deviation αq. The discrete Gaussian distribution $\chi_{\alpha,q}$ is a Gaussian distribution rounded to the nearest integer and reduced modulo q.*

3.3.1.2 Reconciliation method

In a key-exchange protocol, reconciliation refers to the methodology and the collection of procedures utilized by two communicating parties allowing them to arrive at exactly the same keys without having to exchange the key directly. The reconciliation mechanism used by Frodo is a generalised version of the Piekert's key agreement mechanism [Pei14]. Unlike Piekert's mechanism, which extracts a single bit, this allows a larger but fixed number of bits to be extracted. This increases in the number of extracted bits also lead to an increased failure of reconciliation. However, this rate of failure is small for practical applications [BCD+16].

Now we shall give a short description of functions that help in exact key agreement using this new reconciliation method. Let B be the number of bits that one aims to extract from one coefficient in $\mathbb{Z}_q$ be such that $B < (\log_2 q - 1)$ Let $\bar{B} = (\log_2 q - B)$. For any $v \in \mathbb{Z}_q$ the *rounding function* $\lfloor \cdot \rceil_{2^B}$ is defined as

$$\lfloor v \rceil_{2^B} := \lfloor 2^{-\bar{B}} \cdot v \rceil \mod 2^B.$$

This returns the B most significant bits of $(v + 2^{\bar{B}-1}) \mod q$. The *crossround* function $\langle \cdot \rangle_{2^B}$ is defined as

$$\langle v \rangle_{2^B} := \lfloor 2^{-\bar{B}+1} \cdot v \rfloor \mod 2.$$

This function returns the $(B+1)$th most significant bit of v. Now with these two function, we finally define the function Rec. This function is built on the foundations of the reconciliation function proposed by Piekert [Pei14]. The function

works as follows: on input of some $w \in \mathbb{Z}_q$ and a bit $b \in \{0,1\}$, the `Rec` outputs $\lfloor v \rceil_{2^B}$ where v is the closest element to w such that $\langle v \rangle_{2^B} = b$. It is important to note that the bit b acts as a hint for the element w to compute the element v.

Example 3.3.3. *In Figure 3.4, we show a sample example of the functioning of the `Rec` function. The figure shows three adjacent intervals such that the two intervals from the left have the same top B bits however, they differ by their $(B+1)$th bit, while the third interval has the different Bth bit, however the $(B+1)$th bit is the same as the first interval on the left. Now, given the input of the element $w \in \mathbb{Z}_q$ and a hint vector $b = 0$, `Rec` computes the closest vector $v \in \mathbb{Z}_q$ such that $\langle v \rangle_{2^B} = b = 0$.*

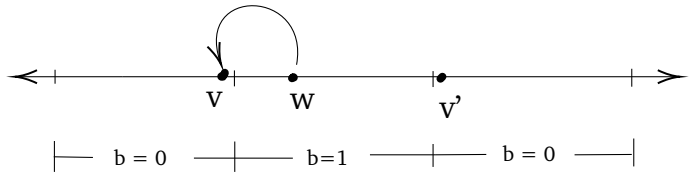

Figure 3.4 – An example of finding closest element with the hint bit b

3.3.1.3 Description of Key Exchange

The key exchange protocol is described in Figure 3.5 works as follows: both Alice and Bob generate the same large matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$. Alice generates two further matrices $\mathbf{S}, \mathbf{E} \in \chi(\mathbb{Z}_q^{n \times \bar{n}})$ and finally computes the public vector

$$\mathbf{B} = \mathbf{AS} + \mathbf{E}.$$

Bob similarly computes two secret matrices $\mathbf{S}', \mathbf{E}' \in \chi(\mathbb{Z}_q^{\bar{m} \times n})$ and another public vector

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

Bob computes the secret matrix $\mathbf{V} \in \mathbb{Z}_q^{\bar{m} \times \bar{n}}$ where $\mathbf{V} = \mathbf{S}'\mathbf{B} + \mathbf{E}''$. Bob then proceeds to compute the shared secret $\mathbf{Key_B} \in \mathbb{Z}_q^{\bar{m} \times \bar{n}}$ by

$$\mathbf{Key_B} := \lfloor \mathbf{V} \rceil_{2^B} = (\lfloor 2^{-\bar{B}} \mathbf{V}_{ij} \rceil \mod 2^B), \quad 1 \le i \le \bar{m}, \ 1 \le j \le \bar{n}.$$

Bob sends the vector $\mathbf{V}$ along with a hint vector $\mathbf{C} = \langle \mathbf{V} \rangle_{2^B}$ where the function $\langle \cdot \rangle_B$ works on each component as

$$\langle v \rangle_{2^B} := \lfloor 2^{-\bar{B}+1} v \rfloor \mod 2.$$

Now for Alice to compute the secret $\mathbf{Key}_A$, she computes the vector $\mathbf{B}'\mathbf{S} \in \mathbb{Z}_q^{\bar{m} \times \bar{n}}$ and takes in the hint vector $\mathbf{C}$. Once this has been computed, Alice then proceeds to compute the nearest vector $\mathbf{W}$ of $\mathbf{B}'\mathbf{S}$ such that for each component,

$$\langle \mathbf{W}_{ij} \rangle_B = \mathbf{C}_{ij}$$

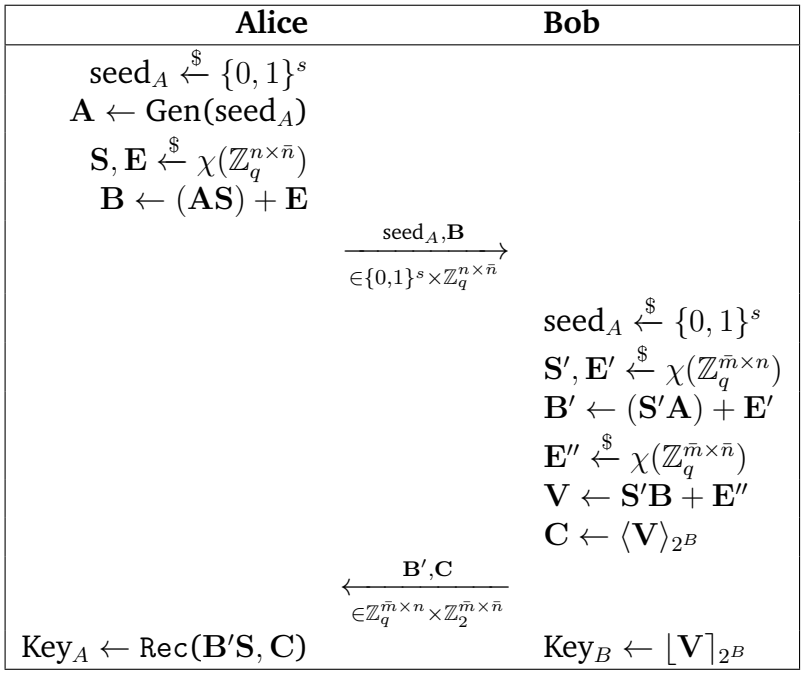

Figure 3.5 – `Frodo` Key-exchange Scheme

Alice finally computes her version of the shared secret $\mathbf{Key}_A$ by

$$\mathbf{Key_A} := \lfloor \mathbf{W} \rceil_B = (\lfloor 2^{-\bar{B}} \mathbf{W}_{ij} \rceil \mod 2^B), \quad 1 \leq i \leq \bar{m}, \ 1 \leq j \leq \bar{n}.$$

The correctness of the basic key reconciliation is based on the fact that the vectors computed by both Alice ($\mathbf{B}'\mathbf{S}$) and Bob ($\mathbf{S}'\mathbf{B} + \mathbf{E}''$) differ only by small amount, thus not influencing their most significant bits.

$$\begin{aligned} \mathbf{B}'\mathbf{S} &= (\mathbf{S}'\mathbf{A} + \mathbf{E}') \cdot \mathbf{S} \\ &= \mathbf{S}' \cdot \mathbf{A} \cdot \mathbf{S} + \mathbf{E}' \cdot \mathbf{S} \\ &= \mathbf{S}' \cdot (\mathbf{AS}) + \mathbf{E}'\mathbf{S} \\ &= \mathbf{S}' \cdot (\mathbf{B} - \mathbf{E}) + \mathbf{E}'\mathbf{S} \\ &= \mathbf{S}'\mathbf{B} - \mathbf{S}'\mathbf{E} + \mathbf{E}'\mathbf{S}. \end{aligned}$$

Thus, the reconciliation holds true if and only if each component of the vector $\mathbf{E}'\mathbf{S} - \mathbf{S}'\mathbf{E} - \mathbf{E}''$ is small than $2^{\log_2 q - B - 2}$ [BCD+16, Claim 3.2].

3.3.1.4 Parameter Choice

There are some key exchange mechanisms, for example [AAB+] and [BCNS15], that use Piekert's key reconciliation method with one bit extraction for making the session key. This was because the number of such reconciliations needed was

far greater than the number of bits required to make the session key. In this case, we have a total of $m \cdot n$ reconciliations. So to achieve a post-quantum security of 128 bits, one requires $mn \cdot B > 256$. With a larger number of bits extracted, one can achieve smaller parameters of m and n, which in turn implies smaller sizes of `LWE` matrices. Finally, we redirect the reader to [BCD+16] for more information and therefore the detailed working of the scheme has been omitted.

3.3.1.5 Failure rate

The probability of reconciliation is 1 when the distance between the two vectors: is less than $q/2^{B+2}$, whereas, when it is larger than $3q/2^{B+2}$, the scheme fails every time [BCD+16]. The success probability decreases from 1 to 0 in between these two extremes. This is described in the figure of example 3.3.2. the `Rec` function can find another vector v' in the interval to the right of the interval containing w. However, it should be noted that this interval does not have the same B most significant bits even though $\langle v' \rangle = b = 0$. Thus the function doesn't choose v'. However for some cases, especially when w lies in the exact middle of the sub-interval, there is a choice to be made, that determines the success of the reconciliation.

Part II

Contribution

Chapter 4

Cryptanalysis of Extension Field Cancellation Cryptosystem

Abstract

We investigate the Extension Field Cancellation (EFC) scheme, which we presented in Section 3.1.4. The organization of the chapter is as follows: first, we present a successful Gröbner basis attack on the first and the second proposed parameters of the scheme. The attack was mounted with complexity much smaller than the claimed security level. We further show that the algebraic system arising from EFC are much easier to solve than a random system of the same size. Briefly, this is due to the apparition of many lower degree equations during Gröbner basis computation. We present a polynomial time method to recover such lower degree relations and also show their usefulness in improving the Gröbner basis attack complexity on EFC.

4.1 Introduction

In this chapter, we investigate a new encryption scheme: Extension Field Cancellation (EFC) [SDP16] which is based on using high degree polynomials over an extension field as a part of public-key construction. We presented the working of this scheme briefly in Section 3.1.4. As seen before, on one hand, most of the multivariate encryption schemes proposed have been subjected to cryptanalysis, on the other hand, EFC has stood so far with no cryptanalysis yet.

4.1.1 Main Results and Organization

This chapter is organized as follows. Section 4.2.2 reports the results of a hybrid Gröbner basis attack [BFP09] on all three challenge parameters of EFC. Using

this message recovery attack, for the first and the second challenge parameter we recover the hidden secret message in 2^{65} and 2^{77} operations respectively, contrary to the claims of 80 bits of security for these parameter. For the third parameter, although the worst-case hybrid Gröbner basis attack takes 2^{80} operations, in an average case, we expect it to be about 2^{79}. Section 4.2.3 provides experimental evidence of the non-random behavior of the public polynomials of EFC. In Section 4.2.4, we show why $\texttt{EFC}_q(0)$, the scheme without any modifiers, is weak. Particularly, we show that a polynomial-time attack exists on $\texttt{EFC}_q(0)$ and $\texttt{EFC}_q^F(0)$, which has been stated informally in Theorem 1 below:

Theorem 4.1.1 (informal). *Given a public-key $(f_1, \ldots, f_{2n}) \in \mathbb{F}_q^{2n}[x_1, \ldots, x_n]$ and the ciphertext $(c_1, \ldots, c_{2n}) \in \mathbb{F}_q^{2n}$ from an instance of $\texttt{EFC}_q(0)$ or $\texttt{EFC}_q^F(0)$, using Gröbner basis, we can recover the hidden secret message in $\mathcal{O}(n^{3\omega})$ which is polynomial in n and where $2 \leq \omega < 3$ is the linear algebra constant.*

We present the full version of Theorem 4.1.1 as well as the proof in Section 4.2.4. We explain how a degree 3 linear combination of the public-keys of $\texttt{EFC}_q(0)$ or $\texttt{EFC}_q^F(0)$ yield linear equations (see Section 4.2.4 for more details).

We extend this methodology to $\texttt{EFC}_q^-$ as well, where we recover quadratic equations from a high degree (degree ≥ 3) combinations of the public-keys. This has been discussed in some details in Sections 4.2.5, 4.2.6, and 4.2.7. This technique is quite similar to the approach used against HFE [FJ03] and the Hidden Matrix scheme [**?**] schemes where the authors show the public-keys exhibiting some algebraic properties are easier to solve than a random system of quadratic equations of the same sizes. We introduce a new technique of explicitly computing and recovering low-degree relations from the public-keys of $\texttt{EFC}_q^-$. To do so, we consider the initial public-keys and their Frobenius powers. The following Claim 4.1.1 informally describes the basic idea. We refer to Section 4.3 for further details.

Claim 4.1.1 (informal). *Given the public-keys equations for an instance of $\texttt{EFC}_q^-$, we can always find some combinations of the public-keys and their Frobenius powers which produce new low-degree relations.*

Using this technique, we can recover the quadratic relations from degree 3 combinations in 151 minutes for the first challenge parameter and 110 minutes for the second challenge parameter. This computation is polynomial-time in the number of variables. Furthermore, we show that adding these new equations along with the public equations make the Gröbner basis computation much more efficient as well as reducing the time complexity by a huge factor. For instance, in case of $\texttt{EFC}_q^-(2)$ with $n = 75$, adding such intermediate equations reduces the run time of F4 from more than a day to 66.05 seconds to compute the Gröbner basis. Thus, this scheme has structural weaknesses that can be easily exploited by an adversary to recover secret messages and thus making the scheme unsuitable for encryption.

4.2 Algebraic Cryptanalysis of EFC

In this section, we show how Gröbner basis (see Section 2.3) and Gröbner basis algorithms can be utilized to attack the EFC scheme. To break the $\mathtt{EFC}_q$ scheme, as described in section 3.1.4, the underlying problem is to find a solution to the system of equations given by public-key ciphertext pairs. In this section, firstly we present a polynomial time key recovery attack on $\mathtt{EFC}_q(0)$, Secondly, we report the results of a Gröbner basis based message recovery attack on the proposed parameters of EFC. Finally, we show why a system of public-keys arising from EFC is much easier to solve than a random system of algebraic equations with the same size. This is supported by both theoretical and experimental observations.

4.2.1 A Key Recovery Attack

In section 3.2.1.1, we presented Patarin's affine multiple attack [Pat96] on $\mathtt{C}^*$ where due to an algebraic property, one can find a bi-linear relation between the plaintexts and the ciphertexts. $\mathtt{EFC}_q(0)$, like $\mathtt{C}^*$, is vulnerable to a form of linearization attack, which recovers the secret-key equations. Recall that decryption of $\mathtt{EFC}_q(0)$ requires solving a system of linear equations which are derived from Equation (3.3). As said earlier, given the ciphertexts $(c_1, \ldots, c_{2n}) \in \mathbb{F}_q^{2n}$, these decryption equations have an equivalent univariate representation over the extension field $\mathbb{F}_{q^n}$, which is of the following form

$$\beta(\chi)\mathcal{C}_1 - \alpha(\chi)\mathcal{C}_2 = 0,$$

where $\mathcal{C}_1 = \varphi(c_1, \ldots, c_n) \in \mathbb{F}_{q^n}$, $\mathcal{C}_2 = \varphi(c_{n+1}, \ldots, c_{2n}) \in \mathbb{F}_{q^n}$, while $\alpha(\chi) \in \mathbb{F}_{q^n}[\chi]$ and $\beta(\chi) \in \mathbb{F}_{q^n}[\chi]$ are defined as in Equation (3.2). This is a bi-linear relation between the plaintext, χ (since both $\alpha(\chi)$ and $\beta(\chi)$ are linear over the base field $\mathbb{F}_q$) and the ciphertexts $\mathcal{C}_1$ and $\mathcal{C}_2$. So choosing enough plaintexts and corresponding ciphertexts, one can recover the coefficients of $\alpha(\chi)$ and $\beta(\chi)$, which were unknown to the attacker.

Hence, using the public-key $P \in \mathbb{F}_q[x_1, \ldots, x_n]$, we can generate several plaintext-ciphertext pairs. For each pair of plaintext-ciphertext $(x'_1, \ldots, x'_n) \in \mathbb{F}_q^n$, $(y'_1, \ldots, y'_{2n}) \in \mathbb{F}_q^{2n}$ given by $P\ (x'_1, \ldots, x'_n) = (y'_1, \ldots, y'_{2n})$, we can substitute

$$X = \varphi(x'_1, \ldots, x'_n),\ \Upsilon_1 = \varphi(y'_1, \ldots, y'_n),\ \Upsilon_2 = \varphi(y'_n, \ldots, y'_{2n}) \in \mathbb{F}_{q^n},$$

into the following linearization equation

$$\Psi(X, \Upsilon_1, \Upsilon_2) = \left(\sum_{i=0}^{n-1} \mathfrak{g}_i X^{q^i-1}\right) \cdot \Upsilon_2 - \left(\sum_{i=0}^{n-1} \mathfrak{h}_i X^{q^i-1}\right) \cdot \Upsilon_1 = 0,$$

to get a linear equation in $2n$ unknowns $\mathfrak{g}_i, \mathfrak{h}_i \in \mathbb{F}_{q^n}$. Therefore, choosing roughly $2n$ plaintext-ciphertext pairs, we can very likely solve the resulting system for the unknown coefficients in $\mathcal{O}(n^3)$ time complexity. Once we have the resulting linear

system of equations, for a given ciphertext $(c_1, \ldots, c_{2n}) \in \mathbb{F}_q^{2n}$, we can recover the actual secret message by solving the univariate equation $\Psi(X) = 0$, which over the base field $\mathbb{F}_q$ represents a system of n linear equations in n variables $(x_1, \ldots, x_n)$. Solving this system has $\mathcal{O}(n^3)$ complexity. This attack was known to the authors of [SDP16]. This idea of recovering the secret-key from the public-keys to obtain the hidden message can also be realised using Gröbner bases. This has been presented in great detail in Section 4.2.4.

For the case of $a > 0$, an attacker needs to guess the correct ciphertext for each of the a public-keys for every choice of $2n$ plaintexts such that these $2n$ plaintext-ciphertext pairs solve for the unknown coefficients of the linear equation Ψ. In worst-case, the complexity is $\mathcal{O}(q^{2na})$, which is exponential in n and the number of missing public-keys, a. Therefore, choosing appropriate value of the parameter a can foil such a key-recovery attack using linearization techniques on $\texttt{EFC}_q^-(a)$.

4.2.2 A Message Recovery Attack

In this part, we show that the challenge parameters of EFC (see Table 3.2) can be attacked thanks to the *hybrid Gröbner basis algorithm* [BFP09, Bet11] which we discussed in Section 2.4. This attack combines both exhaustive search as well as Gröbner basis computation by fixing k out of the n variables and then computing Gröbner basis. The complete set of solutions is recovered from the computation of all q^k Gröbner bases. The attack relies on the idea that the cost of computing Gröbner basis decreases when the ratio between the number of equations and the number of variables increases [BFSY05]. Table 4.1 lists the number of fixed variables and the expected number of operations of the message recovery attack for EFC parameters. The expected number of operations for the hybrid attack is calculated explicitly and is given by:

$$N_{hyb} = \log_2 \left(t \cdot q^k \cdot 2.93 \times 10^9\right),$$

where t is the time taken for computing a Gröbner basis over the public-keys in which we fix k variables. In our experiments, we take this time t to be the average time over 100 runs. The number of expected operations, N_{hyb}, is not an asymptotic estimate as we do not use the expected degree of regularity for computing the complexity of the hybrid Gröbner basis attack.

From Table 4.1, we see that the first parameter $\texttt{EFC}_2^-(10)$ with $n = 83$ is broken just by using 2^{65} bit operations, while the second parameter $\texttt{EFC}_2^{F^-}(8)$ with $n = 83$ is broken by using 2^{77} bit operations. We have not been able to find an attack with number of operations strictly less than 2^{80} on the third challenge parameter of $\texttt{EFC}_3^-(6)$ with $n = 56$, however, fixing 25 variables gives us a sharp upper bound of 2^{80}. Therefore, if this parameter is used, in the worst case instance, the hybrid attack will take 2^{80} bit operations, however, in average case, we expect the number of operations to be about 2^{79}. This is simply because, the total number

of steps required to search for every possible combination is given by $3^{25}(3^{25}+1)/2$. Therefore, the average search time for any combination is simply $(3^{25}+1)/2 \approx 2^{39}$. This result leads us to estimate 2^{79} to be the number of operations taken by the hybrid Gröbner basis attack on an average when the third parameter is chosen. In Table 4.1 we also list the degree of regularity for a semi-regular system, $\mathbf{D}_{SR}$ with the same parameters. The time (Time) and the memory (Mem) column represent the time and space required for one instance of running the Gröbner basis computation. We use the `Magma` [BCP97] 2.19 implementation of `F4` algorithm on a quad-core Intel(R) Xeon(R) CPU E7-4820 v4 2.93 GHz computer with 1 Tb of memory, which is facilitated by the `GroebnerBasis` function.

Parameter	n	k	Time (sec)	Mem (Gb)	$\mathbf{D}_{SR}$	Bit Comp.
$\mathtt{EFC}_2^-(10)$	83	18	48773	115.03	9	65
$\mathtt{EFC}_2^{F-}(8)$	83	39	265	1.719	9	77
$\mathtt{EFC}_3^-(6)$	56	25	667	0.489	10	80

Table 4.1 – Hybrid Gröbner basis attack on `EFC` parameters.

In Figure 4.1, we show the complexity of the hybrid Gröbner basis attack to recover the secret message with respect to increasing value of the number of variables (n) and fixing various fraction of variables (Frac) for $a = 10$. It clearly shows that increasing the value of n will not help as still the hybrid Gröbner basis attack recovers the secret message in time complexity less than 2^{80}. Even if n or a is increased to the extent that algebraic attack complexity is more than 2^{80}, this will sufficiently increase the decryption time, thus making the scheme unsuitable for use.

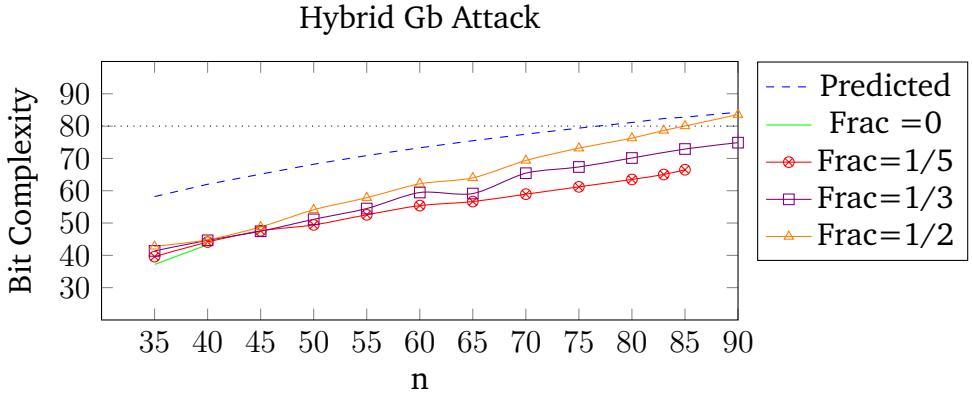

Figure 4.1 – Complexity of hybrid Gröbner Basis attack for $\mathtt{EFC}_2^-(10)$ and various fraction of variables fixed.

In the following sections, we show why EFC is easier to solve than a random system of algebraic equations. We show with experimental evidence that the degree of regularities observed for $\texttt{EFC}_q^-$ show non-random behavior. Also, we explain the apparition of many quadratic or linear polynomials during the computation of a Gröbner basis of the ideal generated by the public polynomials of EFC.

4.2.3 Lower Degree of Regularity

In this section, we provide the first experimental evidence of a bound on the degree of regularity of EFC which is much less than that of semi-regular degree of regularity bound. Figure 4.2 depicts how the maximal degree reached during Gröbner basis computation varies with an increasing value of n and also how it varies with the parameter a, the number of equations removed from the public-keys.

In our experimentation, we also observe that the maximal degree of computation is actually smaller than that was used by the designers of EFC to derive their parameters [SDP16]. The theoretical degree of regularity $\mathbf{D}_{Theo}$ in [SDP16] is given by

$$\mathbf{D}_{Theo} \leq \frac{(q-1)(r+a)}{2} + 2. \tag{4.1}$$

It is important to note that [SDP16] do mention this degree to be an upper bound, but on the other hand, consider this degree to fix their challenge parameters. We represent this as theoretical degree of regularity in Figure 4.2 depicted as $\mathbf{D}_{Theo}$. As seen in the Figure 4.2, the graph of $\mathbf{D}_{Theo}$ grows more than the graph of $\mathbf{D}_{Obs}$ which is the observed degree of regularity for EFC during the experiments. The complexity of an algebraic attack varies exponentially with the degree of regularity (refer Chapter 2). Hence even a drop in this degree by a value of 1 can decrease the bit-security level of the parameters by a significant number of bits.

4.2.4 Analysis of the $\texttt{EFC}_q(0)$ and $\texttt{EFC}_q^F(0)$ instances

Recall from Section 3.1.4, the set of possible solutions for Equation (3.3) is expected to be small [SDP16]. The number of solutions obtained depends on the rank of the linear system of equation (3.3). Over $\mathbb{F}_q$, if the rank of the above system is $n - r$ for $r > 0$, this implies that the kernel of this linear map, is a subspace of co-dimension r in $\mathbb{F}_q^n$. Therefore there are q^r solutions in $\mathbb{F}_q^n$.

If r is close to n then the scheme is not efficient as the decryption then involves pruning through all solutions in the set by computing their corresponding ciphertext and matching it amounting to

$$q^r \cdot \mathcal{O}(n^3) = \mathcal{O}(n^{\zeta+3}), \quad \zeta = \frac{r}{\log_q n}$$

operations. In practice we found the rank of the system of linear equations from equation (3.3) is $(n-1)$ in more than half of the cases while it is $(n-2)$ for almost

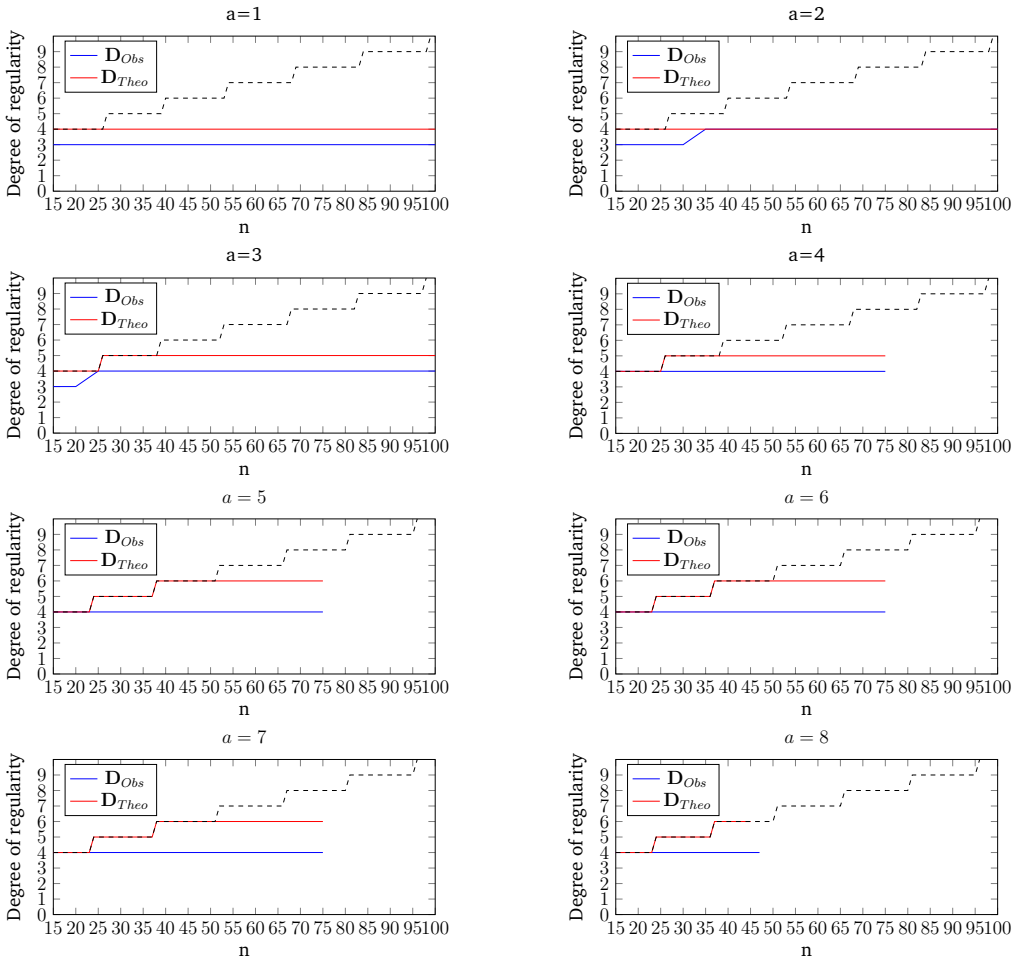

Figure 4.2 – Degree of regularity observed in experiments over $\mathbb{F}_2$, expected by [SDP16] and degree of semi-regularity denoted by black dash.

0.4% of the cases (see Table 4.2), irrespective of value of n. Hence it is safe to assume that for all of the cases, $r \leq 4$.

Rank	$n-1$	$n-2$	$n-3$	$n-4$	$n-5$
%	0.5725	0.3904	0.0361	0.001	0.000

Rank	$n-6$	$n-7$
%	0.000	0.000

Table 4.2 – Percentage of cases where the corresponding rank is observed. Experimentation data from 10000 runs on $n = 20, 30, 45$.

Assumption 4.2.1. *The size of the solution set to the system of linear equations is upper bounded by q^4 and $n > q^{4/3}$.*

We say that EFC scheme is "**well defined**" if Assumption 4.2.1 holds true and the decryption takes polynomial time complexity.

We now show how we can recover this equation (3.3) using Gröbner basis in polynomial time. Additionally, we say there is a *degree drop* when a linear combination of two polynomials $f, g \in \mathbb{F}_q[x_1, \ldots, x_n]$ of degree d produces another polynomial $h(\neq 0) \in \mathbb{F}_q[x_1, \ldots, x_n]$ whose degree $d' < d$.

Theorem 4.2.2. *Given public-keys polynomials $(f_1, \ldots, f_{2n}) \in \mathbb{F}_q^{2n}[x_1, \ldots, x_n]$ and ciphertexts $(c_1, \ldots, c_{2n}) \in \mathbb{F}_q^{2n}$ from a "well defined" instance of $\mathtt{EFC}_q(0)$ or $\mathtt{EFC}_q^F(0)$, we can recover the hidden secret message in polynomial-time complexity of $\mathcal{O}(n^{3\omega})$ where $2 \leq \omega < 3$ is the linear algebra constant.*

Proof. The central map of $\mathtt{EFC}_q(0)$ is given by two n-dimensional vectors of polynomials $F_1, F_2 \in \mathbb{F}_q^n[x_1, \ldots, x_n]$. Let $\mathbf{C_1}, \mathbf{C_2} \in \mathbb{F}_q^n$ be the ciphertexts vectors for F_1 and F_2 respectively. We consider the ideal generated by the following:

$$I = \langle F_1 - \mathbf{C_1}, F_2 - \mathbf{C_2} \rangle . \tag{4.2}$$

We have $F_1 = \alpha_m(\mathbf{x})\mathbf{x}, F_2 = \beta_m(\mathbf{x})\mathbf{x}$ where $\mathbf{x}$ is the vector $(x_1, \ldots, x_n)$. By construction, we have

$$\beta_m(\mathbf{x}) \cdot (F_1 - \mathbf{C_1}) = \beta_m(\mathbf{x})\alpha_m(\mathbf{x})\mathbf{x} - \beta_m(\mathbf{x})\mathbf{C_1},$$

$$\alpha_m(\mathbf{x}) \cdot (F_2 - \mathbf{C_2}) = \alpha_m(\mathbf{x})\beta_m(\mathbf{x})\mathbf{x} - \alpha_m(\mathbf{x})\mathbf{C_2}.$$

Subtracting the above two equations, we obtain $\beta_m(\mathbf{x})\mathbf{C_1} - \alpha_m(\mathbf{x})\mathbf{C_2}$ which is a sequence of n linear polynomials, which are also in the ideal I. As we have a well defined instance of EFC, it implies that Gaussian elimination produces n linear equations in $\mathbb{F}_q[x_1, \ldots, x_n]$.

We show that these linear equations also appear during the Gröbner basis computation. In this case, one proceeds by generating Macaulay matrix (Definition 2.3.22) of degree 3 from the $2n$ quadratic public-key equations in $\mathbb{F}_q[x_1, \ldots, x_n]$. These $2n$ public-keys can also be represented by the set $\{\alpha_m(\mathbf{x})\mathbf{x} - \mathbf{C_1}\} \cup \{\beta_m(\mathbf{x})\mathbf{x} - \mathbf{C_2}\}$. Now each row of the matrix $\alpha_m(\mathbf{x}) \cdot (F_2 - \mathbf{C_2})$ has the following form:

$$\left(\sum_{1 \leq i,j \leq n} a_{ij} x_i \cdot (\beta_m(\mathbf{x})\mathbf{x} - \mathbf{C_2})_j \right) \quad , a_{ij} \in \mathbb{F}_q. \tag{4.3}$$

Similarly each row of $\beta_m(\mathbf{x}) \cdot (\alpha_m(\mathbf{x})\mathbf{x} - \mathbf{C_1})$ has the form

$$\left(\sum_{1 \leq i,j \leq n} b_{ij} x_i \cdot (\alpha_m(\mathbf{x})\mathbf{x} - \mathbf{C_1})_j \right) \quad , b_{ij} \in \mathbb{F}_q. \tag{4.4}$$

Macaulay matrix of degree 3 for the ideal I also has rows of the form

$$\left(\sum_{1 \leq i \leq n} \sum_{1 \leq j \leq 2n} \gamma_{ij} x_i \cdot (f_j - c_j) \right), \tag{4.5}$$

where $(f_j - c_j) \in \{F_1 - \mathbf{C_1} \cup F_2 - \mathbf{C_2}\}$. Hence, $\alpha_m(\mathbf{x}) \cdot (\beta_m(\mathbf{x})\mathbf{x} - \mathbf{C_2})$ and $\beta_m(\mathbf{x}) \cdot (\alpha_m(\mathbf{x})\mathbf{x} - \mathbf{C_1})$, each represent vectors of n cubic polynomials which occur in the Macaulay matrix of degree 3. Consequently, $\beta_m(\mathbf{x})\mathbf{C_1} - \alpha_m(\mathbf{x})\mathbf{C_2}$ also appear during computation. These n linear equations, given by $\beta_m(\mathbf{x})\mathbf{C_1} - \alpha_m(\mathbf{x})\mathbf{C_2}$, are the same set of n linear equations present in the decryption module (refer (3.3)).

During decryption with the original decryption module if there is exactly one solution (i.e. the system is full rank), then in exactly one step of Gröbner basis computation using Macaulay matrix at degree 3, we are able to get n linearly independent linear equations from which we recover the secret. And thus the degree of regularity, $\mathbf{D}$ is 3. We can therefore recover the hidden secret in $\mathcal{O}(n^{3\omega})$.

Now consider the case when the linear equations in the original decryption module is not of full rank, i.e. let's say we have $n - r$ linearly independent linear equations with $r > 0$. As already stated earlier, the n linear equations represented by $\beta_m(\mathbf{x})\mathbf{C_1} - \alpha_m(\mathbf{x})\mathbf{C_2}$ occur in the Macaulay matrix of degree 3 and these equations are exactly the same set of linear equations from equation (3.3), therefore the n linear equations of $\beta_m(\mathbf{x})\mathbf{C_1} - \alpha_m(\mathbf{x})\mathbf{C_2}$ also have rank $n-r$ and produce the same set of solutions as we derive from the decryption process. Hence to find the hidden secret message, we need to prune this solution set of dimension q^r which has a complexity of $\mathcal{O}(n^{3+\zeta})$, where ζ is previously defined in Section 2.2. According to Assumption 4.2.1, we have $\zeta \leq \frac{4}{\log_q n}$ and $n > q^{4/3}$, thus the total complexity of computing the hidden secret is

$$\mathcal{O}(n^{3\omega}) + \mathcal{O}(n^{3+\zeta}) \approx \mathcal{O}(n^{3\omega}).$$

For $\texttt{EFC}_q^F(0)$, each central map polynomial $\alpha(\chi)\chi \in \mathbb{F}_{q^n}[\chi]$ and $\beta(\chi)\chi \in \mathbb{F}_{q^n}[\chi]$ have an extra quadratic term $\beta^3(\chi)$ and $\alpha^3(\chi)$ added to them respectively (while working over a finite field of characteristic two). Addition of these terms doesn't change the structure of the polynomials in the Macaulay matrix of degree 3. Let $\alpha'(\mathbf{x}), \beta'(\mathbf{x}) \in \mathbb{F}_q^n$ be the equivalent base field vector representation of $\alpha^3(\chi)$ and $\beta^3(\chi)$. Therefore, left multiplication of $\alpha'(\mathbf{x})$ by $\alpha_m(\mathbf{x}) \in \mathbb{F}_q^{n\times n}[x_1, \ldots, x_n]$ is an equivalent base field representation of $\alpha(\chi)\alpha^3(\chi) \in \mathbb{F}_{q^n}[\chi]$, which over the base field of order $q = 2$ represents a linear system of equations in $\mathbb{F}_q[x_1, \ldots, x_n]$.

Hence like $\texttt{EFC}_q(0)$, for $\texttt{EFC}_q^F(0)$, each row of the matrices $\alpha_m(\mathbf{x}) \cdot (F_2 - \mathbf{C_2})$ and $\beta_m(\mathbf{x}) \cdot (F_1 - \mathbf{C_1})$ can be written as,

$$\begin{aligned} &\left(\sum_{1\leq i,j\leq n} a_{ij}x_i \cdot ((\beta_m(\mathbf{x})\mathbf{x} + \alpha'(\mathbf{x}) - \mathbf{C_2})_j \right)_k \\ &\left(\sum_{1\leq i,j\leq n} b_{ij}x_i \cdot ((\alpha_m(\mathbf{x})\mathbf{x} + \beta'(\mathbf{x}) - \mathbf{C_1})_j \right)_k \end{aligned} \tag{4.6}$$

for $1 \leq k \leq n$. The polynomials from equation (4.6) represent vectors of $2n$ polynomials and these occur in the Macaulay matrix of degree 3. This implies the polynomials from the following

$$\alpha_m(\mathbf{x})\mathbf{C_1} - \beta_m(\mathbf{x})\mathbf{C_2} - \alpha_m(\mathbf{x})\alpha'(\mathbf{x}) + \beta_m(\mathbf{x})\beta'(\mathbf{x}),$$

also appear during the Gröbner basis computation which represents a system of n linear equations and are the same linear equations in the decryption module of the scheme with the Frobenius modifier. Thus similar to the previous case, we can thus recover the hidden secret by solving these n linear equations with high probability. Hence, some combination of the central map polynomials again yields a system of linear equations observed in the degree 3 Macaulay matrix. Therefore, we conclude that we have a polynomial-time, $\mathcal{O}(n^{3\omega})$, message recovery attack on $\texttt{EFC}_q(0)$ and $\texttt{EFC}_q^F(0)$. □

In Table 4.3, we list the time (average time taken over 5000 Gröbner basis computations using F4 with the same parameters) and the experimental maximal degree observed (denoted by $\mathbf{D}$) for different values of $15 \leq n \leq 100$ during the Gröbner basis computation over the public-keys of $\texttt{EFC}_q(0)$ and $\texttt{EFC}_q^F(0)$. As can been seen, the behavior of the public-keys is unlike a random system of quadratic equations, in which case the degree increases linearly with the number of variables, as depicted in column Semi-reg of Table 4.3. Therefore, we have a experimental proof of the existence of such a polynomial-time attack over the public-keys of $\texttt{EFC}_q(0)$ and $\texttt{EFC}_q^F(0)$.

4.2.5 Extending to $\texttt{EFC}_q^-(a)$

As seen earlier, $\texttt{EFC}_q(0)$ can be broken in polynomial-time by a direct Gröbner basis attack. However, in the case of $\texttt{EFC}_q^-(a)$ we do not necessarily obtain linear

n	$\texttt{EFC}_q(0)$		$\texttt{EFC}_q^F(0)$		$\mathbf{D}_{SR}$
	D	Time	**D**	Time	
15	3	0.00	3	0.00	4
20	3	0.00	3	0.00	4
25	3	0.00	3	0.00	4
30	3	0.01	3	0.01	5
35	3	0.02	3	0.02	5
40	3	0.02	3	0.03	5
45	3	0.04	3	0.04	6
50	3	0.05	3	0.06	6
55	3	0.09	3	0.09	7
60	3	0.12	3	0.13	7
65	3	0.18	3	0.20	7
70	3	0.27	3	0.28	8
75	3	0.35	3	0.35	8
80	3	0.5	3	0.51	8
90	3	0.88	3	0.9	9
100	3	1.33	3	1.37	10

Table 4.3 – Maximal degree observed for Gröbner basis computation in F4 on Magma for $\texttt{EFC}_q(0)$ and $\texttt{EFC}_q^F(0)$. Column $\mathbf{D}_{SR}$ represents the degree of regularity for a random (semi-regular) system of polynomials with same parameters.

equations from the public-keys directly at degree 3 as in the case of $\texttt{EFC}_q(0)$ (Section 4.2.4). Nevertheless, we show that many lower degree equations (generally quadratic) can be computed from combinations of the public-keys at higher degree (at degree 3, 4 or 5 depending on the characteristic of the base field and the parameter a). Before we proceed, we derive an equivalent EFC representation of the $\texttt{EFC}^-$ system for the central map. EFC uses two HFE like polynomials as a part of the central map construction. Vates and Smith-Tone in [VST17] propose a technique of converting a $\texttt{HFE}^-$ scheme into an equivalent HFE system representation. Using the same idea we can now build an equivalent EFC system for $\texttt{EFC}^-$.

Definition 4.2.3 (Embedded forgetting map). *Let $\ell, a \in \mathbb{N}$ and $\mathbb{F}_q$ be a finite field. We call $\phi_a : \mathbb{F}_q^\ell \mapsto \mathbb{F}_q^\ell$ an embedded forgetting map if it maps a vector $\mathbf{v} \in \mathbb{F}_q^\ell$ to another vector $\mathbf{v}' \in \mathbb{F}_q^\ell$ such that*

$$\mathbf{v}'_\mathbf{i} = \begin{cases} \mathbf{v_i} & \text{if } 1 \leq i \leq \ell - a, \\ 0 & \text{if } \ell - a + 1 \leq i \leq \ell. \end{cases}$$

This ϕ_a can be written as a composition of a forgetting map, $\mathbb{F}_q^\ell \mapsto \mathbb{F}_q^{\ell-a}$, forgetting the last a vectors and an embedding map $\mathbb{F}_q^{\ell-a} \hookrightarrow \mathbb{F}_q^\ell$, appending a zeros

to a $(\ell - a)$-dimensional vector. Hence, for clarity of notation, we shall denote such a embedding map ϕ_a as follows

$$\phi_a : \mathbb{F}_q^{\ell} \mapsto \mathbb{F}_q^{\ell-a} \hookrightarrow \mathbb{F}_q^{\ell}.$$

The minus modifier of EFC is formed by removing 'a' polynomials from the public-keys. So $P^- = (p_1, \ldots, p_{2n-a}) = E' \circ T \circ F \circ S \in \mathbb{F}_q^{2n-a}[x_1, \ldots, x_n]$, where $E' : \mathbb{F}_q^{2n} \mapsto \mathbb{F}_q^{2n-a}$ and $S \in \mathbb{F}_q^{n \times n}$, $T \in \mathbb{F}_q^{2n \times 2n}$ are invertible linear transformations. We can now define $\phi_a : \mathbb{F}_q^{2n} \mapsto \mathbb{F}_q^{2n-a} \hookrightarrow \mathbb{F}_q^{2n}$, an embedded forgetting map such that it $\phi_a(G) = (E'(G), 0, \ldots, 0)$ for any $G \in \mathbb{F}_q^{2n}[x_1, \ldots, x_n]$. Thus, building a new system $P^0 \in \mathbb{F}_q^{2n}[x_1, \ldots, x_n]$ by appending zero polynomials to P^-, we get $P^0 = (p_1, \ldots, p_{2n-a}, 0, \ldots, 0) = \phi_a \circ T \circ F \circ S$. One should note that the map ϕ_a maps a vector in $\mathbb{F}_q^{2n}$ to another vector in $\mathbb{F}_q^{2n}$. This is a linear map and can also be written as a matrix $\Phi_a \in \mathbb{F}_q^{2n \times 2n}$. Thus for an embedded forgetting map ϕ_a, we shall interchangeably use its matrix representation Φ_a.

Before proceeding further, we recall that the central map F of EFC (Section 3.1.4) is composed of two polynomial sets

$$F_1 = \varphi^{-1}(\alpha(\chi)\chi),$$

$$F_2 := \varphi^{-1}(\beta(\chi)\chi),$$

where $\varphi : \mathbb{F}_{q^n} \mapsto \mathbb{F}_q^n$ is the natural isomorphism as defined in Section 3.1.4.

Proposition 4.2.4. *Let $F_1, F_2 \in \mathbb{F}_q^n[x_1, \ldots, x_n]$ represent the central map polynomials of $\mathtt{EFC}_q$ and $T \in \mathbb{F}_q^{2n \times 2n}$ be the linear transformation that composes with the central map $F \in \mathbb{F}_q^{2n}[x_1, \ldots, x_n]$ to form the public-key of EFC. Suppose there is a embedded forgetting map $\phi_a : \mathbb{F}_q^{2n} \mapsto \mathbb{F}_q^{2n-a} \hookrightarrow \mathbb{F}_q^{2n}$. Then for the public-keys of $\mathtt{EFC}^-$, there is an equivalent representation of the linear transformation $\Phi_a \circ T$ using two distinct embedded forgetting maps $\phi_{a_1} : \mathbb{F}_q^n \mapsto \mathbb{F}_q^{n-a_1} \hookrightarrow \mathbb{F}_q^n$ and $\phi_{a_2} : \mathbb{F}_q^n \mapsto \mathbb{F}_q^{n-a_2} \hookrightarrow \mathbb{F}_q^n$ such that $a_1 + a_2 = a$ and ϕ_{a_1} acts in composition with F_1 while ϕ_{a_2} composes with F_2 of the central map, where $\circ$ is the composition map.*

Proof. The composition $\Phi_a \circ T \in \mathbb{F}_q^{2n \times 2n}$ represents a $2n \times 2n$ matrix. This matrix has a co-rank of a. Now the rows of the this matrix which on composition with the central map $F \in \mathbb{F}_q^{2n}$ produces a vector of polynomials of which a polynomials are zero polynomials.

Now consider the first n rows of $\Phi_a \circ T \circ F \in \mathtt{F}_q^{2n}[x_1, \ldots, x_n]$. We have supposed that out of these n rows a_1 rows are zero rows. This can be represented as a composition of a $n \times n$ matrix Φ_{a_1}, comprising the same exact a_1 zero rows, with $F_1 \in \mathbb{F}_q^n[x_1, \ldots, x_n]$ giving out the same equations as the first n rows of $\Phi_a \circ T \circ F$.

Similarly take the last n rows of $\Phi_a \circ T \circ F$. We can have another $n \times n$ matrix Φ_{a_2} with a_2 zero rows which on composition with $F_2 \in \mathbb{F}_q^n[x_1, \ldots, x_n]$ results in the same last n rows of $\Phi_a \circ T \circ F$. □

Recall that the public-key is made from the composition of two invertible linear transformations $S \in \mathbb{F}_q^{n\times n}$ and $T \in \mathbb{F}_q^{m\times m}$ along with the central map polynomial F. So, composition $T \circ F$ can be written as

$$T \circ F = \begin{bmatrix} T_1 & T_2 \\ T_3 & T_4 \end{bmatrix} \cdot \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} = \begin{bmatrix} I_n & 0 \\ 0 & I_n \end{bmatrix} \cdot \begin{bmatrix} F_1' \\ F_2' \end{bmatrix} = I_{2n} \cdot \begin{bmatrix} F_1' \\ F_2' \end{bmatrix},$$

where $F_1' = T_1 \cdot F_1 + T_2 \cdot F_2 \in \mathbb{F}_q^n[x_1, \ldots, x_n]$ and $F_2' = T_3 \cdot F_1 + T_4 \cdot F_2 \in \mathbb{F}_q^n[x_1, \ldots, x_n]$. These new polynomials F_1' and F_2' are linear combinations of the previous central map polynomials. Hence, without loss of generality, from now we consider the linear transformation T as I_{2n}, where I_{2n} is an identity matrix of size $2n$.

Lemma 4.2.5. *Let $\Phi_a \in \mathbb{F}_q^{2n\times 2n}$ be a linear transformation of co-rank 'a'. Also let $T \in \mathbb{F}_q^{2n\times 2n}$ be a linear transformation that composes with the central map polynomials $(F_1, F_2) \in \mathbb{F}_q^{2n}[x_1, \ldots, x_n]$. Using Proposition 4.2.4, consider there exists equivalent forgetting maps, $\Phi_{a_1} \in \mathbb{F}_q^{n\times n}$ and $\Phi_{a_2} \in \mathbb{F}_q^{n\times n}$. Also consider, the linear transformation $T \in \mathbb{F}_q^{2n\times 2n}$ to be the identity matrix. There exist a non-singular linear transformation $U \in \mathbb{F}_q^{2n\times 2n}$ and polynomials $\pi_1, \pi_2 \in \mathbb{F}_{q^n}[X]$ of degrees q^{a_1} and q^{a_2} respectively, such that $a_1 + a_2 = a$ and $\Phi_a \circ T = \Phi_a \circ I_{2n} = U \circ (\varphi^{-1}, \varphi^{-1}) \circ (\pi_1, \pi_2) \circ (\varphi, \varphi)$, where I_{2n} is the identity matrix, $\varphi : \mathbb{F}_q^n \mapsto \mathbb{F}_{q^n}$ and the composition function $\circ$ works component wise.*

Proof. As stated earlier, we shall consider the linear transformation T to be the identity matrix I_{2n}. Using Proposition 4.2.4, the linear transformation $\Phi_a \circ T = \Phi_a \circ I_{2n} \in \mathbb{F}_q^{2n\times 2n}$ can be considered as collection of two separate embedded forgetting maps, $\Phi_{a_1}, \Phi_{a_2} \in \mathbb{F}_q^{n\times n}$, each acting on the first n, $F_1 \in \mathbb{F}_q^n[x_1 \ldots, x_n]$ and last n polynomials, $F_2 \in \mathbb{F}_q^n[x_1 \ldots, x_n]$ of the central map, respectively. Suppose we have a_1 polynomials removed from F_1 and a_2 removed from F_2. Thus, we have $a = a_1 + a_2$. Let $V_1 \in \mathbb{F}_{q^n}$ be the kernel of $\Phi_{a_1} \circ I_n \in \mathbb{F}_q^{n\times n}$ and similarly $V_2 \in \mathbb{F}_{q^n}$ be the kernel of $\Phi_{a_2} \circ I_n \in \mathbb{F}_q^{n\times n}$. Let $\pi_1 \in \mathbb{F}_{q^n}[X]$ be the minimal polynomial of the algebraic set V_1 and $\pi_2 \in \mathbb{F}_{q^n}[X]$ be the minimal polynomial for V_2. Now removing a_1 polynomials implies that nullity of V_1 is q^{a_1} and similarly $|V_2| = q^{a_2}$. Thus π_1 and π_2 have degrees q^{a_1} and q^{a_2} respectively and are of the form

$$\pi_1 = \sum_{i=0}^{a_1} c_i X^{q^i}, \ \pi_2 = \sum_{i=0}^{a_2} c_i' X^{q^i},$$

where $c_i, c_i' \in \mathbb{F}_{q^n}$ Taking the same approach as Vates and Smith-Tone [VST17, Lemma 1], we argue that there exists linear transformations $U_1, U_2 \in \mathbb{F}_q^{n\times n}$ such that

$$\Phi_{a_1} \circ I = U_1 \circ \varphi^{-1} \circ \pi_1 \circ \varphi, \ \Phi_{a_2} \circ I = U_2 \circ \varphi^{-1} \circ \pi_2 \circ \varphi. \tag{4.7}$$

Using (D.1), we have

$$\Phi_a \circ I_{2n} = \begin{bmatrix} \Phi_{a_1} \circ I_n \\ \Phi_{a_2} \circ I_n \end{bmatrix} = \begin{bmatrix} U_1 \circ \varphi^{-1} \circ \pi_1 \circ \varphi \\ U_2 \circ \varphi^{-1} \circ \pi_2 \circ \varphi \end{bmatrix} = \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix} \cdot \begin{bmatrix} \varphi^{-1} \circ \pi_1 \circ \varphi \\ \varphi^{-1} \circ \pi_2 \circ \varphi \end{bmatrix}.$$

This above matrix representation can be also written as

$$\Phi_a \circ I_{2n} = U \circ (\varphi^{-1}, \varphi^{-1}) \circ (\pi_1, \pi_2) \circ (\varphi, \varphi),$$

where $U = \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix}$. □

We can now prove the following result:

Proposition 4.2.6. *Let $\mathcal{F}_1 \in \mathbb{F}_{q^n}[\chi]$ and $\mathcal{F}_2 \in \mathbb{F}_{q^n}[\chi]$ be the central map polynomials of $\texttt{EFC}_q(0)$ in the extension field representation. Let there be two polynomials $\pi_1, \pi_2 \in \mathbb{F}_{q^n}[X]$ of degrees q^{a_1} and q^{a_2} respectively, as defined in Lemma 4.2.5. Additionally, we also have $a = a_1 + a_2$. Then the central map polynomials $\mathcal{F}'_1, \mathcal{F}'_2 \in \mathbb{F}_{q^n}[\chi]$ for an instance of $\texttt{EFC}^-_q(a)$ can be written as*

$$\mathcal{F}'_1 = \pi_1 \circ \mathcal{F}_1 = \sum_{i=0}^{a_1} c_i(\alpha(\chi)\chi)^{q^i} \quad , \quad \mathcal{F}'_2 = \pi_2 \circ \mathcal{F}_2 = \sum_{i=0}^{a_2} c'_i(\beta(\chi)\chi)^{q^i}. \tag{4.8}$$

Proof. The public-keys for an instance of $\texttt{EFC}^-_q$ is derived from the following composition of maps

$$P^- = \Phi_a \circ T \circ (\varphi^{-1}, \varphi^{-1}) \circ (\mathcal{F}_1, \mathcal{F}_2) \circ (\varphi, \varphi) \circ S, \tag{4.9}$$

where $\Phi_a, T \in \mathbb{F}_q^{2n \times 2n}$ and $S \in \mathbb{F}_q^{n \times n}$ are linear transformations, and the central map comprises of the two quadratic polynomials $\mathcal{F}_1, \mathcal{F}_2 \in \mathbb{F}_{q^n}[\chi]$. Assuming $T = I_{2n}$ to be the identity matrix, from Lemma 4.2.5, we have

$$\Phi_a \circ I_{2n} = U \circ (\varphi^{-1}, \varphi^{-1}) \circ (\pi_1, \pi_2) \circ (\varphi, \varphi).$$

Thus, replacing this in Equation (4.9), we have

$$\begin{aligned} P^- &= \Phi_a \circ I_{2n} \circ (\varphi^{-1}, \varphi^{-1}) \circ (\mathcal{F}_1, \mathcal{F}_2) \circ (\varphi, \varphi) \circ S \\ &= U \circ (\varphi^{-1}, \varphi^{-1}) \circ (\pi_1, \pi_2) \circ (\varphi, \varphi) \circ (\varphi^{-1}, \varphi^{-1}) \circ (\mathcal{F}_1, \mathcal{F}_2) \circ (\varphi, \varphi) \circ S \\ &= U \circ (\varphi^{-1}, \varphi^{-1}) \circ (\pi_1 \circ \mathcal{F}_1, \pi_2 \circ \mathcal{F}_2) \circ (\varphi, \varphi) \circ S \end{aligned}$$

This is an EFC equivalent public-key representation of the public-keys of $\texttt{EFC}^-$, whose central map polynomials are given by $\pi_1 \circ \mathcal{F}_1$ and $\pi_2 \circ \mathcal{F}_2$. □

From here on we shall denote the ciphertexts from the evaluation of the central map polynomials $\mathcal{F}'_1, \mathcal{F}'_2 \in \mathbb{F}_{q^n}[\chi]$ over some secret message as $\mathcal{C}_1 \in \mathbb{F}_{q^n}$ and $\mathcal{C}_2 \in \mathbb{F}_{q^n}$ respectively.

Now that we have built an extension field representation of $\texttt{EFC}^-$ central map polynomials, we show how we can recover lower degree relations from these equations. First, we illustrate our approach for small values of a. For ease of notation, from now on we shall denote the polynomial $\alpha(\chi)$ as α and similarly $\beta(\chi)$ as β. Hence the product $\alpha(\chi)\chi$ has been represented as $\alpha\chi$ (and $\beta\chi$ for $\beta(\chi)\chi$ similarly).

Now that we have build an equivalent EFC key representation of $\texttt{EFC}^-$, we show using examples how for various values of a, we can recover lower degree relations.

4.2.6 Analysis on the case $\mathtt{EFC}_2^-(1)$

Now we look at the behavior of the public-keys with one equation removed over $\mathbb{F}_2[x_1, \ldots, x_n]$ and analyze the Gröbner basis over these equations. Before proceeding, we revisit the definition of a Frobenius power of a polynomial.

Definition 4.2.7. *For any element* $\mathbf{z}$ *of a finite field* $\mathbb{F}_q$ *of some prime characteristic* p*, the Frobenius function maps* $\mathbf{z}$ *to its* p^{th} *power. Extending this to commutative ring, i.e* $\mathbb{F}_q[\chi]$*, the Frobenius morphism maps a polynomial* $g \in \mathbb{F}_q[\chi]$ *to* g^p*, where* q *is some positive power of a prime* p*.*

For $a = 1$, the central map polynomials (using Equation 4.8) can be written as

$$\mathcal{F}'_1 = \alpha^2\chi^2 + c_0\alpha\chi,$$

$$\mathcal{F}'_2 = \beta\chi.$$

As said earlier, we shall denote the evaluations of the polynomials over a secret message as $\mathcal{C}_1 \in \mathbb{F}_{q^n}$ and $\mathcal{C}_2 \in \mathbb{F}_{q^n}$ respectively. We write two polynomials $\mathcal{H}_1$ and $\mathcal{H}_2$ in $\mathbb{F}_{q^n}[\chi]$ as

$$\mathcal{H}_\mathbf{1} : \mathcal{F}'_1 - \mathcal{C}_1,$$
$$\mathcal{H}_\mathbf{2} : \mathcal{F}'_2 - \mathcal{C}_2.$$

For simplicity let us consider $c_0 = 1$. As shown below, we are able to find 4 different combinations of $\mathcal{H}_1$ and $\mathcal{H}_2$ and their Frobenius powers $\mathcal{H}_1^2, \mathcal{H}_2^2, \mathcal{H}_1^4, \mathcal{H}_2^4$, which represent degree fall from degree 3 to 2. The 4 distinct equations over the extension field $\mathbb{F}_{2^n}$ represent $4n$ quadratic equations in the base field $\mathbb{F}_2$ are as follows:

1. $\mathcal{H}_1 \cdot \beta - \mathcal{H}_2 \cdot \alpha^2\chi - \mathcal{H}_2 \cdot \alpha$
2. $\mathcal{H}_1 \cdot \beta^2 - \mathcal{H}_2^2 \cdot \alpha^2 - \mathcal{H}_2 \cdot \alpha\beta$
3. $(\mathcal{H}_1^2 \cdot \beta^4 - \mathcal{H}_2^4 \cdot \alpha^4) \cdot \chi^2 - \mathcal{H}_2^4 \cdot \alpha^2$
4. $(\mathcal{H}_1^2 \cdot \beta^4 - \mathcal{H}_2^4 \cdot \alpha^4 - \mathcal{H}_1 \cdot \beta^4) \cdot \chi - \mathcal{H}_2^2 \cdot \alpha\beta^2$

Experimental evidence of a direct Gröbner basis attack on the public-key equations shows exactly $4n$ quadratic equations (highlighted in bold) at step degree 3, as can be seen in Table 4.4. For $n = 50$, at the first step degree (SD) 3, we observe 200 quadratic equations while the same for $n = 75$ exhibits 300 quadratic equations. It is also interesting to note that for such small value of $a = 1$, we can recover the secret message from in the very next step of the Gröbner basis computation at degree 3, implying such instances of $\mathtt{EFC}_2^-(1)$ are weak. We also note in both the cases, the first step of F4 computes new quadratic polynomials at degree 2, for instance, for $n = 50$ we observe 93 new quadratic polynomials, while we have 142 new quadratic polynomials for $n = 75$. This number is always less than $2n - 1$, which is the number of public-keys. One can explain such behavior

by the system of public-keys not being full rank, i.e., one can find some public-key equation using a linear combination of the other equations.

Step-degree	$n = 50$, $a = 1$
2	Deg 2: 93
3	**Deg 2: 200** ,Deg 3:830
3	Deg 1: 50

Step-degree	$n = 75$, $a = 1$
2	Deg 2: 142
3	**Deg 2: 300** ,Deg 3:1566
3	Deg 1: 75

Table 4.4 – Number of new polynomials observed during Gröbner basis computation over the public-keys for parameters $n = 50$, $a = 1$, $q = 2$ and $n = 75$, $a = 1$, $q = 2$.

4.2.7 Analysis on the case $\mathtt{EFC}_2^-(2)$

For $a = 2$, there are two sub-cases: $a_1 = 1, a_2 = 1$ and $a_1 = 2, a_2 = 0$. First let us consider the sub-case, $a_1 = 1, a_2 = 1$. So the we have the following polynomials

$$\mathcal{H}_\mathbf{1} : \alpha^2\chi^2 + \alpha\chi - \mathcal{C}_1,$$

$$\mathcal{H}_\mathbf{2} : \beta^2\chi^2 + c_1\beta\chi - \mathcal{C}_2.$$

Again we see that taking the following degree 3 combination of the public-keys gives a degree 2 polynomial.

$$\mathcal{H}_1(\beta^2\chi + \beta) + \mathcal{H}_2(\alpha^2\chi + \alpha).$$

Now consider the other sub-case $a_1 = 2, a_2 = 0$ where we have the following polynomials

$$\mathcal{H}_\mathbf{1} : \alpha^4\chi^4 + \alpha^2\chi^2 + \alpha\chi - \mathcal{C}_1,$$

$$\mathcal{H}_\mathbf{2} : \beta\chi - \mathcal{C}_2.$$

The following degree 3 combination of $\mathcal{H}_1, \mathcal{H}_2$ and $\mathcal{H}_2^2$,

$$\beta^2\mathcal{H}_1 + (\alpha^4\chi^2 + \alpha^2)\mathcal{H}_2^2 + \alpha\beta\mathcal{H}_2,$$

is a quadratic polynomial. Thus this represents a degree drop. Thus we see that, from the structure of the public-keys for $a = 1$ or $a = 2$, we can explicitly recover algebraic combinations of the public-polynomials leading to a degree drop. Table 4.5 gives the experimental evidence of the existence of such lower degree

polynomials for $n = 45$ and $n = 50$. As seen earlier, for $a = 2$, we have two further sub-cases, i.e. $(a_1 = 2, a_2 = 0)$ and $(a_1 = 1, a_2 = 1)$, hence for ease of notation, we shall denote these sub-cases as (a_1, a_2) replaced by their values. For example, $a = 2$ with $a_1 = 1$ and $a_2 = 1$ is denoted as $(1, 1)$. From the table, we observe that for the case of $(1, 1)$ the experiments clearly show the existence of $3n$ quadratic equations at the first step degree 3. Thus, if we are able to find combinations of the public-keys and their Frobenius powers that represent a degree drop from degree 3 to degree 2 over the base field, we can easily recover these above $3n$ quadratic equations.

Step-degree	$n = 45$, $a = 2$ $(1, 1)$
3	**Deg 2: 135** ,Deg 3: 733
3	Deg 3 :3223
4	Deg1 : 45

Step-degree	$n = 50$, $a = 2$ $(1, 1)$
3	**Deg 2: 150**, Deg 3: 865
3	Deg 3:3778
4	Deg 1:50

Step-degree	$n = 45$, $a = 2$ $(2, 0)$
3	**Deg 2: 54**, Deg 3: 733
3	**Deg 2 : 180**, Deg 3:2683
2	Deg 2: 28
3	Deg 1: 44 ,Deg 2 :1

Step-degree	$n = 50$, $a = 2$ $(2, 0)$
3	**Deg 2: 150**, Deg 3: 892
3	**Deg 2 : 200**, Deg 3:3265
2	Deg 2: 30
3	Deg 1: 49, Deg 2 :1

Table 4.5 – Number of new polynomials observed during Gröbner basis computation over the public-keys for $\texttt{EFC}_2^-(2)$ parameters with $n = \{45, 50\}$, and cases $\{(1, 1), (2, 0)\}$.

4.2.8 Analysis on the case $\texttt{EFC}_3^-(1)$ and $\texttt{EFC}_3^-(2)$

Similar to the instance of $q = 2$, while working over a finite field of characteristic 3, the $\texttt{EFC}^-$ public-keys are vulnerable to this attack. We can again write the minus modified central map polynomials for $\texttt{EFC}_3^-(a)$ in an extension field equivalent key

representation. In particular, for $a = 1$, with $a_1 = 1, a_2 = 0$, we have

$$\mathcal{H}_{\mathbf{1}} : \alpha^3\chi^3 + \alpha\chi - \mathcal{C}_1,$$

$$\mathcal{H}_{\mathbf{2}} : \beta\chi - \mathcal{C}_2.$$

Using the same approach of taking combinations of the Frobenius powers of the public-keys over the extension field, we show that there exists multiple systems of quadratic equations from combinations at degree 4

1. $\beta\mathcal{H}_1 - (\alpha^3\beta\chi^2 + \alpha\beta - \alpha^3\chi\mathcal{C}_2)\mathcal{H}_2,$
2. $(\beta^3\mathcal{H}_1 - \alpha^3\mathcal{H}_2^3 - \alpha\beta^2\mathcal{H}_2)\chi - \mathcal{C}_2\alpha\beta\mathcal{H}_2.$

Table 4.6 show apparition of quadratic polynomials at step degree 4 for $n = \{10, 20, 30\}$ for $a = 1$ during Gröbner basis computation in `Magma` using `F4`. It is an interesting observation that the number of low-degree quadratic equations is not linear in the number of variables n, as was the case in even characteristic. According to our estimate, there is a quadratic relationship with the number of variables.

Step-degree	$n = 10$, $a = 1$
2	Deg 2: 14
3	Deg 3:65
4	**Deg2 : 36** , Deg 4 :11
3	Deg 1: 10

Step-degree	$n = 20$, $a = 1$
2	Deg 2: 34
3	Deg 3:200
4	**Deg 2:171**, Deg 4:896
3	Deg 1: 20

Step-degree	$n = 30$, $a = 1$
2	Deg 2: 54
3	Deg 3:380
4	**Deg 2:406**, Deg 4:2254
3	Deg 1: 30

Table 4.6 – Number of new polynomials observed during Gröbner basis computation over the public-keys for $\texttt{EFC}_3^-(1)$ parameters with $n = \{10, 20, 30\}$.

Similarly, for $a = 2$, we have

$$\mathcal{H}_{\mathbf{1}} : \alpha^3\chi^3 + \alpha\chi - \mathcal{C}_1,$$

$$\mathcal{H}_{\mathbf{2}} : \beta^3\chi^3 + \beta\chi - \mathcal{C}_2.$$

One can always find the following equation representing quadratic polynomials (over the base field $\mathbb{F}_q^n$) from the following combination of $\mathcal{H}_1$ and $\mathcal{H}_2$ such that it represents a degree drop from degree 4 to degree 2.

$$\left(\beta^3 \cdot \mathcal{H}_1 - \alpha^3\mathcal{H}_2\right) \cdot \chi^3 - (\alpha\chi - \mathcal{C}_1) \cdot \mathcal{H}_2 - (\beta\chi - \mathcal{C}_2) \cdot \mathcal{H}_1.$$

Clear evidence of such quadratic equations can be found in a Gröbner basis computation of $\texttt{EFC}_3^-(2)$ instance. For example, at step degree 4, with $n = 20$, we find 97 quadratic equations, while $n = 30$ yields 376 quadratic equations. In case of $n = 40$, one observes 741 such quadratic equations at step degree 4. For $a = 2$, one can also find cubic equations at degree 5. At degree 5, the following combination represents a degree 3 polynomial:

$$(\alpha^3\chi^2 + \alpha)\mathcal{H}_2 - (\beta^3\chi^2 + \beta)\mathcal{H}_1.$$

Such equations are observed only when $n > 40$. However, for small n, experimentally these equations are not observed as the Gröbner basis computation doesn't need to proceed beyond degree 4. Thus in this case of $q = 3$, we observe some quadratic equations (at degree 4) and cubic equations (at degree 5), which is unlike the behavior of EFC over $q = 2$. Thus, over an odd characteristic finite field, although the scheme exhibits a higher degree of regularity, the scheme still suffers from the same vulnerability of recoverable intermediate low-degree equations.

A common observation in the previous sections is that as a increases, it becomes increasingly difficult to compute such combinations of public-keys which represent degree fall equations theoretically. This is mainly because increasing the value of a increases the degree of the corresponding public-key equations in their equivalent key representation over the extension field $\mathbb{F}_{q^n}$. However, we now present an algorithm which can discover such equations for any value of a and discuss in great details in the following section.

4.3 A Method to Find Degree Fall Equations

Previously, we theoretically showed how various combinations of the pubic keys at degree 3,4 and 5 produces lower degree polynomials for $\texttt{EFC}_q^-(a)$ and $\texttt{EFC}_{qF}^-(a)$. In this section, we show an explicit method of recovering such combinations of the public polynomials and their Frobenius powers for any parameter choice of number of variables n and number of removed public polynomials 'a' over any finite field $\mathbb{F}_q$.

Claim 4.3.1. *Given the equivalent extension field representation of the public-keys $\mathcal{P}_1$, $\mathcal{P}_2 \in \mathbb{F}_{q^n}[x_1, \ldots, x_n]$ and ciphertexts $\mathcal{C}_1$, $\mathcal{C}_2 \in \mathbb{F}_{q^n}$ of $\texttt{EFC}^-$, let $\mathcal{H}_1 = \mathcal{P}_1 - \mathcal{C}_1 \in \mathbb{F}_{q^n}[x_1, \ldots, x_n]$ and $\mathcal{H}_2 = \mathcal{P}_2 - \mathcal{C}_2 \in \mathbb{F}_{q^n}[x_1, \ldots, x_n]$. We can always find some combination of $\mathcal{H}_1, \mathcal{H}_2$ and their Frobenius powers which produce lower degree relations in $I_{\leq 3}$ ($I_{\leq 4}$ or $I_{\leq 5}$ if $q = 3$) where I is the ideal generated by $\mathcal{H}_1$ and $\mathcal{H}_2$.*

Let us consider the case of $\texttt{EFC}_2^-(a)$. Denote the multiplicands of $\mathcal{H}_1$ and its Frobenius powers $\mathcal{H}_1^q, \ldots, \mathcal{H}_1^{q^{n-1}}$ with p_i. Similarly for $\mathcal{H}_2$ and its Frobenius powers $\mathcal{H}_2^q, \ldots, \mathcal{H}_2^{q^{n-1}}$ we denote the multiplicands with g_i. So at degree 3, any element $\mathcal{T}$

in the ideal I can be represented as

$$\mathcal{T} = p_1 \cdot \mathcal{H}_1 + \cdots + p_{n-1} \cdot \mathcal{H}_1^{q^{n-1}} + g_1 \cdot \mathcal{H}_2 + \cdots + g_{n-1} \cdot \mathcal{H}_2^{q^{n-1}} \tag{4.10}$$

where for all i, $\deg(p_i), \deg(g_i)$ are of the form 2^k with $k \geq 1$. The polynomials p_i and g_i are of the following form

$$p_i = \sum_{j=0}^{\lfloor \log_q \mathfrak{D}_1 \rfloor} \mathfrak{p}_{j+1,i} \chi^{q^j} + \mathfrak{p}_{i0}, \quad g_i = \sum_{j=0}^{\lfloor \log_q \mathfrak{D}_2 \rfloor} \mathfrak{g}_{j+1,i} \chi^{q^j} + \mathfrak{g}_{i0} \tag{4.11}$$

where $\mathfrak{D}_1 = \mathfrak{D}_2 = q^{n-1}$. The coefficients of $\mathfrak{p}_{j+1,i}$, $\mathfrak{p}_{i0}$, $\mathfrak{g}_{j+1,i}$ and $\mathfrak{g}_{i0}$'s are unknown. Now to show that quadratic polynomials occur in the ideal $I_{\leq 3}$, let us consider that $\mathcal{T}$ represents a degree 2 polynomial in the base field $\mathbb{F}_q$. Hence we consider all the coefficients in $\mathcal{T}$, whose monomial are of the form $\chi^{q^i+q^j+q^k}$, i.e the monomials of degree 3, where $(i \neq j \neq k)$. Thus the coefficients of these cubic monomials are all 0. As a result we get an over-determined system of $\binom{n}{3}$ linear equations in $2n^2 + 2n$ unknowns. This system of equations is consistent since there exists a trivial solution of all zeros.

`KernelMatrix` function in Magma allows us to do Gaussian elimination and compute the solution set. The solution set is returned as a basis matrix. The number of solutions is related to the rank of this matrix. We have not been able to show theoretically prove that the rank of the above system is less than the number of variables. However, experimental observations show that the rank is equal to $7n$ for $\texttt{EFC}_2^-(a)$, which is strictly less than number of unknowns, $(2n^2 + 2n)$. Thus we have a polynomial time process to recover non-zero lower degree relations from the combination of the public-keys and their Frobenius powers.

A similar approach can be applied for an `EFC` instance in odd characteristic. Especially for the case of $q = 3$, one can find some combination of $\mathcal{H}_1$ and $\mathcal{H}_2$ and their Frobenius powers in their degree 4 truncated ideal which produces quadratic equations. We construct this combination exactly in the same way as earlier (see equation 4.10). However, now we consider polynomials p_i and g_i of the following form

$$p_i = \mathfrak{p}_{i0} + \sum_{j=0}^{\lfloor \log_q \mathfrak{D}_1 \rfloor} \mathfrak{p}_{j+1,i} \chi^{q^j} + \sum_{j=0}^{\lfloor \log_q \mathfrak{D}_1 \rfloor} \sum_{k=0}^{j} \mathfrak{p}_{j+1,k+1,i} \chi^{q^j+q^k},$$

$$g_i = \mathfrak{g}_{i0} + \sum_{j=0}^{\lfloor \log_q \mathfrak{D}_2 \rfloor} \mathfrak{g}_{j+1,i} \chi^{q^j} + \sum_{j=0}^{\lfloor \log_q \mathfrak{D}_2 \rfloor} \sum_{k=0}^{j} \mathfrak{g}_{j+1,k+1,i} \chi^{q^j+q^k}$$

where $\mathfrak{D}_1 = \mathfrak{D}_2 = q^{n-1} + q^{n-2}$. We consider all the coefficients of the polynomial $\mathcal{T}$ whose monomials are of the form $\chi^{q^i+q^j+q^k}$ and $\chi^{q^i+q^j+q^k+q^l}$, i.e the monomials of degree 3 and 4, where $(i \neq j \neq k \neq l)$. The coefficients of such monomials are all 0 since we consider the polynomial $\mathcal{T}$ to be quadratic. As a result we get an over-determined system of $\binom{n}{3} + \binom{n}{4}$ linear equations in $n^3 + 3n^2 + 2n$ unknowns.

4.3.1 An improvement on the method

As we very well know, the complexity of computing the kernel for this is directly related to the size of the coefficient matrix of the system of linear equations which we derive from the monomials with zero coefficients. This leads to one natural question: whether the size of the matrix can be reduced which can make the computation of the kernel more efficient. Let us look at the structure of the polynomials $\mathcal{H}_1$ and $\mathcal{H}_2$.

$$\mathcal{H}_1 = \sum_{j=0}^{a_1} \left(\sum_{i=0}^{n-1} \mathcal{A}_{ij} \chi^{q^i+1} \right)^{q^j} + \mathcal{A}_0 = \left(\sum_{i=0}^{n-1} \mathcal{A}_{ia_1} \chi^{q^{i+a_1}+q^{a_1}} \right) + \cdots + \left(\sum_{i=0}^{n-1} \mathcal{A}_{i0} \chi^{q^i+1} \right) + \mathcal{A}_0$$

$$\mathcal{H}_2 = \sum_{j=0}^{a_2} \left(\sum_{i=0}^{n-1} \mathcal{B}_{ij} \chi^{q^i+1} \right)^{q^j} + \mathcal{B}_0 = \left(\sum_{i=0}^{n-1} \mathcal{B}_{ia_2} \chi^{q^{i+a_2}+q^{a_2}} \right) + \cdots + \left(\sum_{i=0}^{n-1} \mathcal{B}_{i0} \chi^{q^i+1} \right) + \mathcal{B}_0$$

In the previous section, the idea of constructing the polynomial $\mathcal{T}$ in equation (4.10) is to achieve cancellation of cubic terms by multiplying linear polynomials p_i's and g_i's of the form of equation (4.11). Multiplying p_1 and g_1 to $\mathcal{H}_1$ and $\mathcal{H}_2$ respectively produces cubic monomials of the form $\chi^{q^{i+j}+q^j+q^k}$ with coefficients contributed by $\mathfrak{p}_{j,1}$, $\mathfrak{g}_{j,1}$ and $\mathcal{H}_i$'s. Similarly multiplying p_2 and g_2 to $\mathcal{H}_1^q$ and $\mathcal{H}_2^q$ respectively produces cubic monomials whose coefficients are contributed by $\mathfrak{p}_{j,2}$, $\mathfrak{g}_{j,2}$ and $\mathcal{H}_i^q$'s. However, there are two interesting observations in this case:

1. Either the monomials which have $\mathfrak{p}_{2,2}$ or $\mathfrak{g}_{2,2}$ as coefficients are either linear or quadratic, or
2. The monomials which have the unknowns as coefficients are all cubic or of higher degree over the base field.

To exhibit this we present the following example.

Example 4.3.2. *Let us consider an EFC instance, where $q = 2$, $n = 3$ and $a = 1$. Without loss of generality, we consider that all the coefficients of the central trapdoor maps are all 1. Hence, we have*

$$\mathcal{H}_1 = \sum_{i=0}^{2} \chi^{q^i+1} - c_1 = \chi^2 + \chi^3 + \chi^5 - c_1$$

$$\mathcal{H}_2 = \sum_{i=0}^{2} \chi^{q^{i+1}+q} + \sum_{i=0}^{2} \chi^{q^i+1} - c_2 = \chi^2 + \chi^3 + \chi^4 + \chi^5 + \chi^6 + \chi^{10} - c_2$$

Consider the polynomials p_i's and g_i's are represented as following

$$p_1 = \mathfrak{p}_{1,1} + \mathfrak{p}_{2,1}\chi + \mathfrak{p}_{3,1}\chi^2 + \mathfrak{p}_{4,1}\chi^4, \quad g_1 = \mathfrak{g}_{1,1} + \mathfrak{g}_{2,1}\chi + \mathfrak{g}_{3,1}\chi^2 + \mathfrak{g}_{4,1}\chi^4$$

$$p_2 = \mathfrak{p}_{1,2} + \mathfrak{p}_{2,2}\chi + \mathfrak{p}_{3,2}\chi^2 + \mathfrak{p}_{4,2}\chi^4, \quad g_2 = \mathfrak{g}_{1,2} + \mathfrak{g}_{2,2}\chi + \mathfrak{g}_{3,2}\chi^2 + \mathfrak{g}_{4,2}\chi^4$$
$$p_3 = \mathfrak{p}_{1,3} + \mathfrak{p}_{2,3}\chi + \mathfrak{p}_{3,3}\chi^2 + \mathfrak{p}_{4,3}\chi^4, \quad g_3 = \mathfrak{g}_{1,3} + \mathfrak{g}_{2,3}\chi + \mathfrak{g}_{3,3}\chi^2 + \mathfrak{g}_{4,3}\chi^4$$

Now, in the polynomial $\mathcal{T}$, constructed similar to equation (4.10), the monomials which take contribution from the "variables" $\mathfrak{p}_{2,2}$ and $\mathfrak{g}_{2,2}$ are as follows:

$$\mathfrak{p}_{2,2} \to \chi \cdot (\chi^3 + \chi^4 + \chi^6) = \chi^4 + \chi^5 + \chi^7 = \chi^4 + \chi^5 + 1$$
$$\mathfrak{g}_{2,2} \to \chi \cdot (\chi + \chi^3 + \chi^4 + \chi^5) = \chi^2 + \chi^4 + \chi^5 + \chi^6$$

We observe that none of the monomials involving $\mathfrak{p}_{2,2}$ and $\mathfrak{g}_{2,2}$ are cubic.

We shall recall that we are considering coefficients of all the possible cubic monomials from $\mathcal{T}$. These coefficients form a system of linear equations with $\mathfrak{p}_{i,j}$ and $\mathfrak{g}_{i,j}$ as variables. Hence, ignoring those variables which either do not occur in these linear equations does not effect our process of computing the intermediate equations. Consequently, for constructing the polynomial $\mathcal{T}$, we can ignore the "variables" $\mathfrak{p}_{2,2}$ and $\mathfrak{g}_{2,2}$ from the polynomials p_2 and g_2 respectively. This can be extended for polynomials p_k and g_k. More specifically, for the polynomials p_k one can ignore the variables $\{\mathfrak{p}_{2,k}, \ldots, \mathfrak{p}_{k,k}\}$. Similarly for polynomials g_k, one can ignore the variables $\{\mathfrak{g}_{2,k}, \ldots, \mathfrak{g}_{k,k}\}$. Ignoring such variables, reduces the number of unknowns from $(2n^2 + 2n)$ to $(n^2 + 3n)$. This process has an advantage of smaller dimension of the kernel basis, increasing the probability of recovering a useful intermediate quadratic equation.

Similar to the even characteristic case, in the case of $q = 3$, there are some unknowns which either have no contribution to quadratic monomials (i.e all the monomials involving the unknowns are either cubic or of degree 4) or all the monomials involving the unknowns are quadratic. More specifically, such unknowns do not have monomials which are a mix of both quadratic and cubic or higher. Hence, we can ignore such unknowns from our construction of the quadratic polynomial $\mathcal{T}$. The following example exhibits such a case.

Example 4.3.3. *Let us consider an EFC instance, where $q = 3$, $n = 3$ and $a = 1$. Without loss of generality, we consider that all the coefficients of the central trapdoor maps are all 1. Hence, we have*

$$\mathcal{H}_1 = \sum_{i=0}^{2} \chi^{q^i+1} - c_1 = \chi^2 + \chi^4 + \chi^{10} - c_1$$

$$\mathcal{H}_2 = \sum_{i=0}^{2} \chi^{q^{i+1}+q} + \sum_{i=0}^{2} \chi^{q^i+1} - c_2 = \chi^2 + 2\chi^4 + \chi^6 + \chi^{10} + \chi^{12} - c_2$$

Consider the polynomials p_i's and g_i's are represented as following

$$p_2 = \mathfrak{p}_{1,2} + \mathfrak{p}_{2,2}\chi + \mathfrak{p}_{3,2}\chi^2 + \cdots + \mathfrak{p}_{10,2}\chi^{18}$$

$$g_2 = \mathfrak{g}_{1,2} + \mathfrak{g}_{2,2}\chi + \mathfrak{g}_{3,2}\chi^2 + \cdots + \mathfrak{g}_{10,2}\chi^{18}$$

Now, in the polynomial $\mathcal{T}$, constructed similar to equation (4.10), the monomials which take contribution from the "variables" $\mathfrak{p}_{2,2}$ and $\mathfrak{g}_{2,2}$ are as follows:

$$\mathfrak{p}_{2,2} \to \chi \cdot (\chi^4 + \chi^6 + \chi^{12}) = \chi^5 + \chi^7 + \chi^{13}$$

$$\mathfrak{g}_{2,2} \to \chi \cdot (\chi^4 + \chi^6 + 8\chi^{12} + \chi^{10} + \chi^{18}) = \chi^5 + \chi^7 + 8\chi^{13} + \chi^{11} + \chi^{19}$$

We see that all the monomials which take $\mathfrak{p}_{2,2}$ and $\mathfrak{g}_{2,2}$ are cubic. Since the polynomial $\mathcal{T}$ represents a quadratic polynomial, hence unknowns $\mathfrak{p}_{2,2}$ and $\mathfrak{g}_{2,2}$ have no contribution in quadratic part of the polynomial $\mathcal{T}$. Therefore, in this case, we can ignore such variables.

In this case, we can decrease the number of variables from $n^3 + 3n^2 + 2n$ to

$$\frac{4n^3 + 12n^2 + 20n}{6}$$

by ignoring variables $\{\mathfrak{p}_{2,k}, \ldots, \mathfrak{p}_{(k^2+k)/2,k}\}$ and $\{\mathfrak{g}_{2,k}, \ldots, \mathfrak{g}_{(k^2+k)/2,k}\}$ from polynomials p_k and g_k respectively.

4.4 Are the Degree Fall Equations Useful?

In the previous sections, we showed how we recover the intermediate lower degree equations. Now we shall show how such added intermediate equations are useful to a Gröbner basis computation from an efficiency point of view. It is a well-known fact that having more equations make Gröbner basis computation easier. Sometimes, adding the equations which are useful for the algorithm reduces the maximal degree reached during computation. As mentioned earlier, this maximal degree, also known as the *degree of regularity*, is the key parameter for understanding the complexity of Gröbner basis computation. We have seen that the complexity of computing Gröbner basis is exponential in the degree of regularity, $\mathbf{D}$.
Take the case of $a = a_1 = 1, a_2 = 0$. In Section 4.2.6, we found four relations between the public-key equations which shows a degree drop from 3 to 2. Let us consider one of these equations

$$\mathcal{C}_2(\alpha^2\chi + \alpha) + \mathcal{C}_1\beta = 0. \tag{IR-1}$$

In Table 4.7, we present the experiments for $n = 75$ and $q = 2$ by adding the quadratic equations represented by the equation (IR-1) over the base field $\mathbb{F}_q[x_1, \ldots, x_n]$ to the public-key equations of the corresponding $\mathtt{EFC}_2^-$ system. The left table represents the computation of Gröbner basis with the input of only the public-keys while the right table represents that of public-keys with the new equations (from (IR-1)). Appending these equations helps the Gröbner basis computation as the time for computing the Gröbner basis reduces from 37 seconds for

the left table to 7.5 seconds for the right in Table 4.7. Interestingly, with these additional equations, one complete step degree is skipped yielding the n linear equations in just 2 steps of the F4 algorithm.

Now let us take the case of $a = 2$ such that $a_1 = a_2 = 1$ (i.e. one equation is removed from each of the set $\mathcal{F}_1$ and $\mathcal{F}_2$). In Section 4.2.7, we observed the following combination which represents a degree fall from 3 to 2.

$$\mathcal{C}_2(\alpha^2\chi + \alpha) + \mathcal{C}_1(\beta^2\chi + \beta) = 0. \tag{IR-2}$$

In Table 4.8 we demonstrate the degree and the number of equations at each step of F4 for an $\mathtt{EFC}_2^-(2)$ instance with $n = 75$. On the left table, the maximal degree reached at which linear equations were observed was 4, but as soon as we add the Equation (IR-2) to the public-keys, this maximal degree reduces to 3. So these polynomials are some of the intermediate equations that are observed during the Gröbner basis computation and these are useful to reduce the computation complexity. To compute the Gröbner basis, F4 took 66.05 seconds for Case 2, while for Case 1, F4 took about more than a day. So with a drop in $\mathbf{D}$ from 4 to 3, the complexity reduces significantly. Tables 4.9, 4.10 and 4.11 list the times (in seconds taken average over 5000 runs) taken for message recovery from the public-keys equations using the Gröbner basis approach (represented in the tables as "Pub. Keys") and compare with the time taken for message recovery when the new equations to the Gröbner basis computation (represented as "Pub. Keys + New eqs.") are added. From the tables we observe that with an increase in the number of variables, adding the intermediate low-degree equations becomes much more useful from the Gröbner basis computation point of view. For example, in Table 4.9, for $n = 30$ adding the intermediate equations decreases the time from 0.04 seconds to 0.02 seconds, while for $n = 80$, the decrease by almost 7 times. Similarly in Table 4.10, for $n = 30$ adding the corresponding intermediate equations reduces the time 3 folds while for $n = 60$, the drop in the timings is almost 400 times. This observation is consistent even for the Frobenius case (see Table 4.11), where for $n = 20$ the time decreases from 0.15 seconds to 0.08 seconds while for $n = 50$, the decrease is almost 5 folds.

Step-degree	Public-keys
2	Deg 2: 142
3	Deg 2: 300 ,Deg 3:1566
3	Deg 1: 75

Step-degree	Public-keys+ (IR-1)
2	Deg 2: 216
3	Deg 1: 75

Table 4.7 – Number of new polynomials observed during Gröbner basis computation for $\mathtt{EFC}_2^-(1)$ with $n = 75$.

Step-degree	Public-keys
2	Deg 2: 141
3	Deg 2: 225 ,Deg 3: 1623
3	Deg 3: 6905
4	**Deg 1:75**
Step-degree	Public-keys + (IR-2)
2	Deg 2: 215
3	Deg 2: 237, Deg 3: 2904
2	Deg 2: 35
3	**Deg 1: 75**

Table 4.8 – Number of new polynomials observed during Gröbner basis computation for $\texttt{EFC}_2^-(2)$ with $n = 75$ and $a_1 = a_2 = 1$.

n	PK	PK+(IR-1)
10	0.00	0.00
20	0.02	0.01
30	0.04	0.02
40	0.23	0.03
50	0.65	0.09
60	1.29	0.23
70	2.87	0.44
80	5.37	0.76

Table 4.9 – Time for $\texttt{EFC}_2^-(1)$

n	PK	PK+(IR-2)
10	0.00	0.00
20	0.02	0.02
30	0.25	0.08
40	31.66	0.48
50	387.03	1.68
60	1413.32	3.43

Table 4.10 – Time for $\texttt{EFC}_2^-(2)$

Thus, we see for $\texttt{EFC}_q^-(a)$ how we can obtain equations of a lower degree from the public equations in polynomial-time, which when added along with the public equations make the Gröbner basis computation much more efficient as well as reducing the time complexity by a huge factor. Now that we have shown the underlying weakness of the minus modified scheme, we shall extend these attacks to the challenge parameters.

4.5 Experimental Results and Observations

In this section we present the experimental results when we mount a Gröbner basis attack against EFC and its variants. We show that the equations that we recovered can be useful in certain cases for more efficient Gröbner basis attack.

We take the following example of $n = 40$, $q = 2$ and $a = 5$. A direct Gröbner basis computation exhibit the following behaviour with intermediate low degree

n	PK	PK+ NewEqs
10	0.00	0.00
20	0.15	0.08
30	2.9	0.9
40	34.28	7.8
50	234.55	47.22

Table 4.11 – Time for $\texttt{EFC}_2^{F^-}(1)$

equations observed at step degrees 3 and 4 (see Table 4.12). Especially at step degree (SD) 3, we observe n quadratic equations. Using the original method

	Public-Keys
SD 3	**2: 40, 3: 649**
SD 3	3: 950
SD 4	3: 2570, 4 : 12686
SD 4	1: 40

Table 4.12 – Gröbner Basis computation on $\texttt{EFC}_2^-(5)$ public-keys for $n = 40$ using F4. SD represents the step degree in the F4. The number of polynomials observed of degree are represented as "degree : number"

of section 4.3, explicitly computing the degree drop equations requires solving a system of 9880 linear equations in 3280 unknowns. This takes approximately 64 seconds on a Intel Xeon CPU E7-4820 v4 machine taking a space of approximately 2 Gb. The kernel has a basis of dimension $7n$. Using the improved method of section 4.3.1, we solve 9880 linear equations in 1720 variables which takes 128 seconds and recover the intermediate relations.

4.5.1 Attack on Challenge Parameters

For the first challenge parameter, computing a degree 3 truncated Gröbner basis also shows the presence of $3n$ quadratic equations as seen in Table 4.13. Recovering the intermediate equations involves solving a system of 91881 linear equations over 7138 variables which we were able to solve in $\approx$ 151 minutes using approximately 36 Gb of memory. In Table 4.14 we represent the behavior of Gröbner basis computation when the intermediate equations are added (represented in the table with header "Pub. Keys + New eqs"). Adding 3 good combinations allows the F4 algorithm to not spend time in recovering such polynomials during `Reduction` and `SymbolicProcessing` procedure. In Table 4.14, the tables on the right represent the computation of the Gröbner basis on set of public-keys along with the

intermediate equations, which we represent as "Pub. keys + New eqs", while the table on the left represents the Gröbner basis computation only on the public-keys (represented in the table as "Pub. Keys").

A similar attack on the second parameter can be mounted. Even with the Frobenius modifier, the idea is to determine the intermediate quadratic equations at step degree 3. Especially for the case of $a_1 = 5, a_2 = 3$, we observe $4n$ quadratic equations (see Table 4.13). Recovering the quadratic equations takes 110 minutes and 32 Gb of memory for solving a system of 82160 linear equations in 6880 variables. The dimension of the basis of the kernel is 168. Experimental timings show that adding these intermediate equations reduces the Gröbner basis computation timings from 15 seconds to 1.5 seconds and we can recover all the linear equations in just a single step of `F4` in `Magma`.

As we have already said, direct Gröbner basis attacks were not possible due to limitation of memory, therefore the actual degree of regularity could for the first and the second challenge parameters could not be determined. However, experimental results for parameter values close to first parameter (see Figure 4.2) show that the expected degree of regularity is 5, which is much smaller than that 8, which was assumed while setting the parameter [SDP16]. Therefore, using our technique for all degrees, in the best case, we estimate that the degree of regularity would be improve by 1, taking the complexity of Gröbner basis computation to around 2^{76} (with the choice of $\omega = 3$). Choosing a more practical value for the constant $\omega = 2.37$, the Gröbner basis attack complexity is 2^{60}. Similar experiments for the second challenge parameter show the estimated degree of regularity is 6, as oppose to 8 [SDP16]. Therefore, one can estimate that the complexity of the direct Gröbner basis attack to be approximately, $83^{6\omega} \approx 2^{76}$ (with the choice of $\omega = 2$). As seen for smaller parameters, adding new intermediate relations to the Gröbner basis is expected to reduce the degree of regularity to 5, thus reducing the complexity of the Gröbner basis attack to about 2^{63}. For a more practical choice of the constant $\omega = 2.37$, the complexity is about 2^{75}, which is still less than the security strength claimed by [SDP16].

In this case of the second challenge parameter with $(a_1, a_2) = (5, 3)$, one interesting observation is that the degree of regularity is 3, i.e. adding the Frobenius modifier weakens the `EFC` scheme to some level, however this demands further research. Experiments for the third challenge parameter were not possible because of memory limitations. However, according to our estimates, computing the kernel requires solving 395010 linear equations in 123536 unknowns.

Experiments on $\texttt{EFC}_3^-(6)$ with $n = 20$ show that the degree of regularity is 5 while for $n = 30$, the degree of regularity is at least 6. Therefore, for the third challenge parameter, utilizing such intermediate relations along with the public-keys to compute the Gröbner basis, our estimate for the degree of regularity is 6. Assuming this estimate to be correct, we can estimate the complexity of Gröbner basis computation as $n^{\omega \mathbf{D}} = 59^{12} \approx 2^{71}$ (with the choice of $\omega = 2$). One must

still note that with a choice of $\omega = 2.37$, the complexity is 2^{83}. This complexity is slightly higher than the current computational limit of 2^{80} operations on a classical computer, however, NIST recommends at least 112 bits of security strength [BR11] for any cryptographic algorithm in practice.

$n = 83$, $a = 10$	
SD 2	2: 156
SD 3	**2: 249, 3: 1898**
SD 3	3: 8059

$n = 83$, $a = 8$	
SD 3	**2: 332, 3: 1834**
SD 2	2: 32
SD 3	1: 79

Table 4.13 – Degree 3 truncated Gröbner basis computation on $\mathtt{EFC}_2^-(10)$ public-keys and $\mathtt{EFC}_2^{F^-}(8)$ public-keys with new equations with $n = 83$ using F4.

Step-degree	Pub. Keys
2	2: 156
3	**2: 249, 3: 1898**
3	3: 8059

Step-degree	Pub. Keys + New eqs
2	2: 156
3	**3: 1898**
3	3: 8059

Table 4.14 – Degree 3 truncated Gröbner basis computation on $\mathtt{EFC}_2^-(10)$ public-keys and public-keys with new equations with $n = 83$ using F4.

4.6 Conclusion

In this chapter, we showed that Extension Field Cancellation scheme [SDP16] is vulnerable to algebraic attacks using Gröbner basis techniques. The main results include: first an explanation of the fixed degree of regularity for $\mathtt{EFC}_q(0)$ from its structure, secondly for $\mathtt{EFC}_q^-(a)$ and $\mathtt{EFC}_q^{F^-}(a)$ how we are able to obtain equations of lower degree from the public equations in polynomial time, which when added along with the public equations make the Gröbner basis computation much more efficient as well as reducing the time complexity by a huge factor. Finally we also show that the challenge parameter using hybrid Gröbner basis attack is broken. Thus this scheme has structural weaknesses which can be easily exploited by any adversary to recover secret messages.

Chapter 5

Solving Polynomials with Noise

Abstract

Solving a system of equations ($\texttt{PoSSo}_q$) is already known to be an NP-Hard problem. In this past decade, some research work has been put into understanding the mathematical problem of solving a system of noisy equations. This problem has been shown to be as hard as some of the well known hard problems over lattices. Our goal in this chapter is to provide a latest state of the art on the problem. In addition to that, we provide a survey of all the attacks that are solve this problem of $\texttt{PoSSo}_q$ with noise.

5.1 Motivation

In Chapter 2, we discussed the problem of solving a system of polynomial equations. Solving a system of polynomials with noise is another variant that has appeared in cryptography recently, however, formally, this problem has not been discussed in texts at all. In this chapter, we formalize this problem of solving a system of polynomial equations which are erroneous, which we call as the Polynomial System Solving With Noise (PoSSoWN).

This problem was briefly introduced (in a poster) as *Polynomials With Error* (PWE) problem [ALFP] in 2011. [AFFP11] was the first work which took a step towards formalizing a new class of noisy ideal based problems. Based on these problems, the paper takes a particular focus on a class of multivariate schemes, which are known as "Polly Cracker". Polly Cracker schemes in general are those multivariate schemes whose secret key is a Gröbner basis of a multivariate ideal. In particular, the secret key comprises of the Gröbner basis, say $G \subset \mathcal{P}$, where $\mathcal{P}$ is a polynomial ring. The public key is comprised of a degree bounded system of polynomials $F \subset \langle G \rangle$ and a set of terms, say $\mathcal{T} \subset \mathcal{P}$, which maps to itself bijectively under the normal form map of the Gröber basis G. This set $\mathcal{T}$ therefore comprises

of all the terms which lie below the Gröbner basis staircase. The plaintext space is a vector space spanned by the terms of $\mathcal{T}$. Decryption involves computing a normal form over the ciphertexts, which removes all the terms in the ideal $\langle G \rangle$, yielding the plaintext message. [AFFP11] extended this idea of a scheme to design a secure and somewhat homomorphic Polly Cracker-like encryption primitive based on noisy ideal hard problems. In the same year, Albrecht *et al.* proposed a family of closely related problem known as *Max*-PoSSo which is the problem of finding any vector that satisfies the maximum number of polynomials in the input system [AC11].

Even though these problems have been proposed for quite some time, not much attention has been paid to them since then. The goal of this chapter, is to have a detailed look at the one common underlying problem of all these family of problems discussed above, which is the problem of solving a noisy polynomial system (PoSSoWN). This is unlike [AFFP11], where the goal was to formalize a particular class of hard problems that are suitable for the design of a Polly Cracker scheme. In addition to it, we also present the current state of the art algorithms that solve this problem. The PoSSoWN problem can be defined as follows

Definition 5.1.1 (PoSSoWN)**.** *Let $\mathbb{F}_q$ be a finite field, $P = (f_1, \ldots, f_m) \in \mathbb{F}_q[x_1, \ldots, x_n]^m$ be a system of polynomials and χ be some probability distribution on $\mathbb{F}_q$. The problem of* PoSSoWN *is to find -if any- $(s_1, \ldots, s_n) \in \mathbb{F}_q^n$ such that for all $f_i \in P$, we have $f_i(s_1, \ldots, s_n) = e_i$, where $e_i \in \mathbb{F}_q$ is some error chosen uniformly from the distribution χ.*

5.2 Hardness of the PoSSoWN Problem

In this section, we investigate the hardness of the PoSSoWN problem. We first consider the PoSSoWN problem in the linear case and relate it to the well established LWE problem [Reg09]. Then we deal with the non-linear case of PoSSoWN and relate it to another problem which has been proved to be NP hard.

Notation. We shall write $x := a$ for assigning value a to a variable x, and $x \leftarrow_{\$} \chi$ for sampling x from a set χ. We also denote $\mathbb{F}_q[x_1, \ldots, x_n]$ as a polynomial ring with n variables $(x_1, \ldots, x_n)$ over a finite field $\mathbb{F}_q$. $\mathbb{F}_q[x_1, \ldots, x_n]_{\leq d}$ denotes the set of polynomials in the polynomial ring with degree less than or equal to d. Finally, we call an algorithm to PPT if it runs in probabilistic polynomial time.

In complexity theory, to prove the hardness of any mathematical problem, a technique which is very commonly used is *reduction*. Reduction is an algorithm that allows transformation of a problem to another problem. Game-based formalization of computational problems [BR06] is one method which allows us to perform such

reductions. Game-based reductions conceptualize an adversary's interaction with a problem as a kind of game. The interaction of these games is highlighted by an *advantage* that the adversary has in finding the correct outcome of the game. Now, using such games, if one can find an algorithm that transforms one computational problem to another, we can subsequently compute the advantage an adversary has over one problem with respect to another problem.

Hence, a problem 1 is as hard as problem 2, if an adversary against problem 1 has an advantage which is at most that enjoyed by another adversary against problem 2. Therefore, in this section, we shall use this approach to describe the hardness of PoSSoWN. Every game includes **Initialize** and **Finalize** procedures. The game also has specifications of procedures for responding to an adversary's other oracle queries. For any adversary $\mathcal{A}$, the **Initialize** procedure runs and are passed to $\mathcal{A}$. Other procedures answer to the oracle queries of $\mathcal{A}$. Upon termination of $\mathcal{A}$, the output is passed to **Finalize**, which returns the outcome y of the game. This is denoted by $\texttt{Game}^{\mathcal{A}} \implies y$.

Let us now look at the hardness of the problem of solving a system of noisy equations. Using Definition 5.1.1, we understand the PoSSoWN game as follows:

Game 5.2.1. *The problem of solving a system of equations with noise can be understood through a game* $\texttt{PoSSoWN}_{\mathbb{F}_q,d,\chi}(\lambda)$ *as shown in Figure 5.1. The advantage of a* PPT *algorithm* $\mathcal{A}$ *in solving the* PoSSoWN *problem is defined by*

$$\mathbf{Adv}^{possown}_{\mathbb{F}_q,d,\chi,A}(\lambda) := Pr\left[\texttt{PoSSoWN}^{A}_{\mathbb{F}_q,d,\chi}(\lambda) \implies True\right]$$

Initialize(1^λ)	Sample()	Finalize(s')
begin $n \leftarrow n(\lambda)$ $s \leftarrow_\$ \mathbb{F}_q^n$; **return** $(1^\lambda, n)$; **end**	**begin** $f \leftarrow_\$ \mathbb{F}_q[x_1, \ldots, x_n]_{\leq d}$; $e \leftarrow_\$ \chi$ $f' \leftarrow f(s) + e$; **return** (f', f); **end**	**begin** **return** $(s' = s)$ **end**

Figure 5.1 – Game PoSSoWN

5.2.1 Hardness of PoSSoWN: The Linear Case

We begin with an instance of PoSSoWN, where we sample elements from $\mathbb{F}_q[x_1, \ldots, x_n]$ of degree 1. We also recall the LWE problem, which we defined in Definition 3.3.1, and formalize it into an LWE game.

Game 5.2.2. LWE *is defined through a game* $\texttt{LWE}_{n,q,\chi}$ *as shown in Figure 5.2. The advantage of a* PPT *adversary* $\mathcal{A}$ *in solving the* LWE *problem is defined by*

$$\mathbf{Adv}^{lwe}_{n,q,\chi,A}(\lambda) := Pr\left[\texttt{LWE}^{A}_{n,q,\chi}(\lambda) \implies True\right]$$

Initialize(1^λ)	Sample()	Finalize(s')
begin $n \leftarrow n(\lambda)$ $s \leftarrow_\$ \mathbb{Z}_q^n$; **return** $(1^\lambda, n)$; **end**	**begin** $a \leftarrow_\$ \mathbb{Z}_q^n$; $e \leftarrow_\$ \chi$ $b \leftarrow \sum_i a_i s_i + e$; **return** (a, b); **end**	**begin** **return** $(s' = s)$ **end**

Figure 5.2 – Game $\texttt{LWE}_{n,q,\chi}$

From the definition of LWE (see definition 3.3.1), it is easy to see that there is an equivalence of PoSSoWN when we consider the input system to be a system of linear equations. We formalize this in the following lemma.

Lemma 5.2.3 (LWE hard $\implies$ PoSSoWN hard for $\mathbf{d = 1}$)**.** *Let q be a prime. Then for any PPT adversary $\mathcal{A}$ against the* PoSSoWN *problem, there exists a PPT adversary $\mathcal{B}$ against the* LWE *problem such that*

$$\mathbf{Adv}^{possown}_{\mathbb{F}_q,1,\chi,\mathcal{A}}(\lambda) = \mathbf{Adv}^{lwe}_{n,q,\chi,\mathcal{B}}(\lambda)$$

Proof. We shall construct an adversary $\mathcal{B}$ against the LWE problem based on an adversary $\mathcal{A}$ against the PoSSoWN problem for $b = 1$. Algorithm $\mathcal{B}$ initializes $\mathcal{A}$ with λ. When $\mathcal{A}$ calls its **Sample** procedure, $\mathcal{B}$ queries its own **Sample** oracle to obtain $(a, b) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ where $a = (a_1, \ldots, a_n) \in \mathbb{Z}_q^n$. It returns $\sum_i a_i x_i - b$ to $\mathcal{A}$. This is a valid PoSSoWN sample of degree 1. When $\mathcal{A}$ calls **Finalize** on s, $\mathcal{B}$ also calls **Finalize** on $s = (s_1, \ldots, s_n)$. $\mathcal{B}$ succeeds whenever $\mathcal{A}$ succeeds. We see that for a s from PoSSoWN, $\sum_i a_i x_i - b$ gives $\sum_i a_i s_i = e$, which is a valid LWE sample $(a, \sum_i a_i s_i + e)$. □

5.2.2 Hardness of PoSSoWN: The Non-Linear Case

For degree $d > 1$, the hardness of PoSSoWN problem can be related to an ideal-based hard problem, the Gröbner basis with Noise (GBN) problem. The GBN problem was proposed by Albrecht *et al.* in [AFFP11]. To formally define the problem, we use a few algorithms which we define as follows. The first algorithm is the algorithm to generate a reduced Gröbner basis which we derive directly from Definition 2.3.11. We denote this algorithm as ReduceGB [AFFP11, Algorithm 1] and is represented here in Algorithm 7. Using this algorithm, we proceed to describe another algorithm which generates a Gröbner basis given a polynomial ring defined over a finite field with prime characteristic. We denote this algorithm as $\texttt{GBGen}(1^\lambda, \mathbb{F}_q[x_1, \ldots, x_n], d, \succ)$ [AFFP11, Algorithm 2] and is described in Algorithm 8. The algorithm takes in a polynomial ring $\mathbb{F}_q[x_1, \ldots, x_n]$ and a degree bound d and outputs a reduced Gröbner basis [AFFP11, Lemma 6].

Using the algorithm GBGen, we are now ready to formally describe the GBN problem.

Game 5.2.4. *[AFFP11] The Gröbner basis with noise problem is defined through game* $\texttt{GBN}_{\mathbb{F}_q,n,d,\chi,GBGen(\cdot)}$ *as shown in Figure 5.3. The advantage of a* PPT *adversary* $\mathcal{A}$ *in solving the* GBN *problem is*

$$\mathbf{Adv}^{gbn}_{\mathbb{F}_q,n,d,\chi,\texttt{GBGen},\mathcal{A}}(\lambda) := Pr\left[\texttt{GBN}^{\mathcal{A}}_{\mathbb{F}_q,n,d,\chi,GBGen(\cdot)}(\lambda) \implies True\right]$$

Initialize$(1^\lambda, \mathbb{F}_q[x_1,\ldots,x_n], d)$	**Sample()**	**Finalize**(G')
begin $G \leftarrow_\$ \texttt{GbGen}(1^\lambda, \mathbb{F}_q[x], d)$; **return** $(1^\lambda, \mathbb{F}_q[x])$; **end**	**begin** $f \leftarrow_\$ \mathbb{F}_q[x]_{\leq b}$; $e \leftarrow_\$ \chi$ $f' \leftarrow f - (f \mod G) + e$; **return** f'; **end**	**begin** **return** $(G' = G)$ **end**

Figure 5.3 – Game GBN

Algorithm 7 ReduceGB(G)

$\overline{G} \leftarrow \{\}$
while $G \neq \emptyset$ **do**
  $f \leftarrow$ smallest element of G wrt some ordering
  $G \leftarrow G \backslash \{f\}$;
  if $\texttt{LM}(f) \notin \langle \texttt{LM}(\overline{G}) \rangle$ **then**
    $\overline{G} \leftarrow \bar{G} \cup \{\texttt{LC}(f)^{-1} \cdot f\}$;
  end if
end while
return $[\, h \mod \overline{G} \backslash \{h\} \,|\, h \in \overline{G}]$

The GBN problem has been shown to be at least as hard as the well established LWE problem [AFFP11, Lemma 11, 12]. Now, we proceed with the GBN and relate the PoSSOWN problem to it, which we formalize in the following lemma.

Lemma 5.2.5 (GBN hard $\implies$ PoSSoWN hard)**.** *Let* $\mathbb{F}_q$ *be a finite field. Then for any* PPT *adversary* $\mathcal{A}$ *against the* PoSSoWN *problem, there exists a* PPT *adversary* $\mathcal{B}$ *against the* GBN *problem such that*

$$\mathbf{Adv}^{possown}_{\mathbb{F}_q,d,\chi,\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{gbn}_{\mathbb{F}_q,n,d,\chi,\texttt{GBGen},\mathcal{B}}(\lambda)$$

Proof. Let us construct an adversary $\mathcal{B}$ for the PoSSoWN problem from an adversary $\mathcal{A}$ against the GBN problem. Algorithm $\mathcal{A}$ initializes $\mathcal{B}$ with $s = (s_1,\ldots,s_n) \in \mathbb{F}_q^n$ such that for $\mathcal{A}$, the Gröbner basis is initialized as $\{x_1 - s_1, \ldots, x_n - s_n\} \in \mathbb{F}_q[x_1,\ldots,x_n]$. When $\mathcal{B}$ calls **Sample**, $\mathcal{A}$ queries its own procedure **Sample** to get

Algorithm 8 $\texttt{GBGen}(1^\lambda, \mathbb{F}_q[x_1, \ldots, x_n], d, \succ)$

if d=0 **then return** $\{0\}$
else
  for $i \in \{0, \ldots, n-1\}$ **do**
    $g_i \leftarrow x_i$;
  end for
end if
for $m_j \in M_{\succ, \texttt{LM}(g_i)}$ **do**
  $c_{ij} \leftarrow \mathbb{F}_q$;
  $g_i \leftarrow g_i + c_{ij} m_j$;
end for
return $\texttt{ReduceGB}(\{g_0, \ldots, g_{n-1}\})$

$f' \in \mathbb{F}_q[x_1, \ldots, x_n]$. $\mathcal{B}$ then returns $(f - (f \mod G), f'(s))$ to $\mathcal{A}$. This is a valid PoSSoWN sample. When $\mathcal{A}$ calls **Finalize** on some s', $\mathcal{B}$ calls its own **Finalize** procedure on the basis $G' = \{x_1 - s'_1, \ldots, x_n - s'_n\}$. □

From the previous lemma, we see that there is a polynomial time reduction from the GBN problem to the PoSSoWN problem. Therefore, any algorithm which can solve the GBN problem, can be transformed into an algorithm to solve the PoSSoWN. We can also demonstrate that if one can solve some instances of the PoSSoWN problem, then there exists an algorithm which solves all instances of PoSSoWN. This is generally referred to as the average-case to worst-case reduction for a problem and for PoSSoWN this has been formalized in the following lemma.

Lemma 5.2.6 (Average-case to Worst-case reduction for $\texttt{PoSSoWN}_q$)**.** *Let $\mathcal{A}$ be a PPT adversary against $\texttt{PoSSoWN}_{\mathbb{F}_q, d, \mathcal{A}, \chi}$ that is successful for a fixed polynomial fraction of secrets in $\mathbb{F}_q^n$ with overwhelming probability. Then there exists a PPT adversary $\mathcal{B}$ that solves $\texttt{PoSSoWN}_{\mathbb{F}_q, d, \mathcal{B}, \chi}$ on all possible secrets over $\mathbb{F}_q^n$ with sufficiently high confidence. More precisely, provided we have $\mathbf{Adv}^{possown}_{\mathbb{F}_q, d, \chi, \mathcal{A}} > 1/p(\lambda)$ for some polynomial p, then*

$$\mathbf{Adv}^{possown}_{\mathbb{F}_q, d, \chi, \mathcal{B}} > (1/p - 1/q^n.)$$

Proof. This proof is very similar to the proof of [AFFP11, Lemma 10]. The idea of the proof is to find a class of linear transformations that allow randomization a specific value of a secret $\mathbf{s}$. We can denote a reduced Gröbner basis G_s with respect to the fixed secret value $\mathbf{s} \in \mathbb{F}_q^n$ that is of the form $G_s = \{x_1 - s_1, \ldots, x_n - s_n\} \in \mathbb{F}_q[x_1, \ldots, x_n]$ where $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{F}_q^n$. We denote the degree d truncated ideal generated by G_s as $\mathcal{I}_{s, \leq d} \subset \mathbb{F}_q[x_1, \ldots, x_n]$.

Let χ be an error distribution which samples from $\mathbb{F}_q$. We represent the erroneous ideal $\mathcal{J}_{\mathbf{s}, \chi} = \mathcal{I}_{\mathbf{s}, \leq d} + \chi$. The implication of this notation is that any choice of polynomial $g \in \mathcal{J}_{\mathbf{s}}$ can be written as $g = g' + e$ where $g' \in \mathcal{I}_{\mathbf{s}}$ and $e \leftarrow_\$ \chi$. We consider a linear transformation $\mathcal{L}_{\mathbf{t}} : \mathbb{F}_q[x_1, \ldots, x_n] \mapsto \mathbb{F}_q[x_1, \ldots, x_n]$ such that

$\mathcal{L}_{\mathbf{t}}(f) = f(\mathbf{t})$ where $\mathbf{t} = [x_1 - t_1, \ldots, x_n - t_n]$ with $t_i \in \mathbb{F}_q$. Thus the image of the ideal $\mathcal{I}_{\mathbf{s}}$ under $\mathcal{L}_{\mathbf{t}}$ is denoted by $\mathcal{I}_{\mathbf{s+t}}$, which is also the ideal generated by the basis $G_{\mathbf{s+t}} = [x_1 - s_1 - t_1, \ldots, x_n - s_n - t_n]$. Clearly, there is a one to one correspondence between the ideal $\mathcal{I}_{\mathbf{s}}$ and $\mathcal{I}_{\mathbf{s+t}}$ and the variety of the ideal $\mathcal{I}_{\mathbf{s+t}}$ contains only $\mathbf{s} + \mathbf{t} \in \mathbb{F}_q^n$. Thus, $G_{\mathbf{s+t}}$ is a reduced Gröbner basis for the ideal $\mathcal{I}_{\mathbf{s+t}}$.

Now, for any error distribution χ, the image of the map $\mathcal{J}_{\mathbf{s},\chi}$ under the transformation $\mathcal{L}_{\mathbf{t}}$ is $\mathcal{J}_{\mathbf{s+t},\chi}$. We use $\mathcal{A}$ a polynomial number of times on $\mathcal{L}_{\mathbf{t}}(\mathcal{J}_s)$, each with freshly chosen $\mathbf{t} \leftarrow_{\$} \mathbb{F}_q^n$. Therefore, the adversary $\mathcal{A}$ will output the correct $\mathbf{s} + \mathbf{t}$ at least once with overwhelming probability, from which we can recover $\mathbf{s}$. Now, this verification process is a PPT process. Therefore the success probability of the algorithm $\mathcal{B}$ is either $> 1/p$ or exactly equal to $1/q^n$ (which is the success probability of the adversary randomly choosing the correct secret value). Hence, the advantage of $\mathcal{B}$ is at least $(1/p - 1/q^n)$.

Hence, given any PPT adversarial algorithm $\mathcal{A}$, which solves the PoSSoWN problem only over a polynomial fraction of secrets, one can always find an PPT adversary $\mathcal{B}$ that solve the PoSSoWN problem for any the possible values of the secret over $\mathbb{F}_q$. □

It is not hard to see that the PoSSo problem described in Chapter 2 is a very special instance of the PoSSoWN problem, where the choice of the error for each polynomial in the system of equations is zero. Since, PoSSo is already proven to be a NP-Complete [GJ79], it is not hard to see conclude that PoSSoWN is at least as hard as PoSSo. To prove this more formally, we first present the PoSSo problem (see Section 2.1) as a game based problem like the PoSSoWN problem.

Game 5.2.7. *The problem of solving a system of equations with noise can be understood through a game* $\texttt{PoSSo}_{\mathbb{F}_q,d,\chi}(\lambda)$ *as shown in Figure 5.4. The advantage of a* PPT *algorithm* $\mathcal{A}$ *in solving the* PoSSo *problem is defined by*

$$\mathbf{Adv}^{posso}_{\mathbb{F}_q,d,\chi,A}(\lambda) := Pr\left[\texttt{PoSSo}^{A}_{\mathbb{F}_q,d,\chi}(\lambda) \implies True\right]$$

Initialize(1^λ)	**Sample**()	**Finalize**(s')
begin	**begin**	**begin**
$n \leftarrow n(\lambda)$	$f \leftarrow_{\$} \mathbb{F}_q[x_1, \ldots, x_n]_{\leq d}$;	**return** $(s' = s)$
$s \leftarrow_{\$} \mathbb{F}_q^n$;	$f' \leftarrow f(s)$;	**end**
return $(1^\lambda, n)$;	**return** (f', f);	
end	**end**	

Figure 5.4 – Game PoSSo

Proposition 5.2.8. *PoSSo is NP-Hard implies the problem of PoSSoWN with uniform noise is also NP-Hard.*

Proof. As previously mentioned, the PoSSo is a special case of PoSSoWN where all the errors are chosen to be specifically 0. When the noise is chosen from a uniform distribution, the probability of choosing zero error is same as choosing any other error. Thus, any algorithm which solves the PoSSoWN problem, can solve any instance of PoSSo and there is a polynomial time reduction between these two algorithms. Since PoSSo is NP-Hard, any instance of PoSSoWN with uniform noise is also NP-Hard. □

In the next section, we shall look at some of the algorithms which can solve this PoSSoWN problem.

5.3 Algorithms to Solve PoSSoWN

In the following sections, we discuss the possible methods of solving this problem and thus provide the hardness results, which relates to our problem.

5.3.1 Arora-Ge Gröbner Basis Method

An algorithm to solve the LWE problem with small Gaussian noise was proposed by Sanjeev Arora and Rong Ge in [AG11]. They rely on constructing a higher degree univariate polynomial from a given public key equation, such that it takes into consideration all the possible values of error the noisy public key could have, irrespective of the noise distribution. Consider an LWE instance, where the errors are chosen uniformly from a distribution Ψ_k, which has a range $[-k, k]$ of integers. Additionally, the value of $k \in \mathbb{Z}$ is such that $2k + 1 < q$, i.e., the error is always an integer in the range $(-(q-1)/2, (q-1)/2)$, where q is a prime. The algorithm constructs the following polynomial $P \in \mathbb{Z}_q[x_1, \ldots, x_n]$ such that $P(\eta) = 0$, where η is the error.

$$P(\eta) = \eta \prod_{j=1}^{k} (\eta - j)(\eta + j).$$

This polynomial P is of degree $2k + 1$. Recall from Section 3.3, an LWE instance could be written as a system of equations of the form $b = a \cdot z + e$ where $a, b \in \mathbb{F}_q^n, e \in \Psi_k$ and z is an n-dimensional variable vector. So substituting the error variable η with $a \cdot z + b$ in the polynomial $P(\eta)$, we obtain a degree $2k + 1$ polynomial in the variables $z = (z_1 \ldots, z_n)$. Linearization of the polynomial produces a linear equation from P but over $N = \binom{n+(2k+1)}{n}$ new variables. If we query the LWE oracle $\approx \mathcal{O}(N \log q)$ number of times and apply this above described technique, we obtain a system of linear equations, which one can solve with Gaussian elimination with some high probability. We can apply a similar approach to solve PoSSoWN and thus also holds true for a system of noisy quadratic equations.

An instance of $\texttt{PoSSoWN}_q$ involves equations which are of the form $f_i - b_i = f_i - f_i(\mathbf{s}) - e_i \in \mathbb{F}_q[x_1, \ldots, x_n]$, where $f_i \in \mathbb{F}_q[x_1, \ldots, x_n]$, $\mathbf{s} \in \mathbb{F}_q^n$ and $e_i \in \Psi_k$ is an error.

Alternatively, we can represent the error (η) by the following *error polynomial*:

$$\eta = b - f(\mathbf{x}), \tag{5.1}$$

where $\mathbf{x}$ is vector of variables $(x_1, \ldots, x_n)$. We construct the polynomial $P(\eta) \in \mathbb{F}_q[x_1, \ldots, x_n]$ such that

$$P(\eta) = \eta \prod_{j=1}^{k} (\eta - j)(\eta + j). \tag{5.2}$$

An instance of $\texttt{PoSSoWN}$ comprises of a system $F \in \mathbb{F}_q^m[x_1, \ldots, x_n]$ of m quadratic noisy equations. Therefore, corresponding to each polynomial $f_i \in F$, we can construct a polynomial $P_i \in \mathbb{F}_q[x_1, \ldots, x_n]$ of degree $4k+2$. It is quite intuitive to see that the polynomial P_i equals zero when $\mathbf{x} = \mathbf{s}$. In addition to our system of polynomials 6.6, we have another set of n equations of the form

$$\begin{aligned}
(x_1 + k) \cdots (x_1 + 1) x_1 (x_1 - 1) \cdots (x_1 - k) &= 0, \\
(x_2 + k) \cdots (x_2 + 1) x_2 (x_2 - 1) \cdots (x_2 - k) &= 0, \\
&\vdots \\
(x_n + k) \cdots (x_n + 1) x_n (x_n - 1) \cdots (x_n - k) &= 0.
\end{aligned}$$

So if we are able to find a Gröbner basis of this system of equations along with system $P_i(x) = 0$, then we will recover $\mathbf{s}$.

Recall from Chapter 2, Gröbner basis algorithms such as F4 and F5 are techniques to recover solutions to a system of equations. The complexity of F5 algorithm [Fau02] over a system of m polynomials in $\mathbb{F}_q$ is upper bounded by

$$\mathcal{O}\left(m' D_{reg} \binom{n + D_{reg}}{D_{reg}}^{\omega}\right),$$

where D_{reg} is the degree of regularity of $\langle P_1, P_2, \ldots P_{m'} \rangle$ and $2 \leq \omega < 3$ is the linear algebra constant. Before, we proceed, we present some assumptions, which are important for the results of this section.

Assumption 5.3.1. *For our problem, the instance of $\texttt{PoSSoWN}_q$ consists of a system of noisy quadratic polynomials denoted by $F = (f_1, \ldots, f_m) \in \mathbb{F}_q[x_1, \ldots, x_n]$ where each f_i are of the form*

$$f(x_1, \ldots, x_n) = \sum_{1 \leq i,j \leq n} a_{ij} x_i x_j + \sum_{1 \leq i \leq n} b_i x_i + e,$$

where $a_{ij}, b_i \in \mathbb{F}_q$ and $e \leftarrow_{\$} \Psi_k$.

Assumption 5.3.2. *Let* $(G_1, G_2, \sum_{ij} G_{1_{ij,k}} x_i x_j + \sum_i G_{2_k} x_i + e_k) = (G_1, G_2, \mathbf{c}) \in \mathbb{F}_q^{n^2 \times m} \times \mathbb{F}_q^{n \times m} \times \mathbb{F}_q^m$ *be such that* G_1, G_2 *are sampled uniformly at random and* e *is chosen uniformly from the distribution* Ψ_k*. Let* $P(x)$ *be the polynomial as defined in Equation* (5.2)*. We define*

$$P_1 = P(c_1 - \sum_{ij} G_{1_{ij,1}} x_i x_j - \sum_i G_{2_1} x_i) = 0$$

$$\vdots$$

$$P_m = P(c_m - \sum_{ij} G_{1_{ij,m}} x_i x_j - \sum_i G_{2_m} x_i) = 0.$$

It holds that $\langle P_1, \ldots, P_m \rangle$ *is semi-regular (See Definition 2.3.35).*

Using Assumption 5.3.2, we consider the system of polynomials $\{P_1 \ldots, P_m\}$ as semi-regular. Therefore, its Hilbert polynomial [CLO06] is given by

$$H(z) = \frac{(1 - z^{2k+1})^n (1 - z^d)^m}{(1 - z)^n}, \tag{5.3}$$

where $d = 4k + 2$ and m is the number of available equations and n is the number of variables. The degree of regularity D_{reg} is given by the index of the first non-positive coefficient in the expansion of the Hilbert polynomial (Equation 5.3).

Note 5.3.1. *The results of complexity in this section rely majorly on the assumption that the system of equations occurring from the Arora-Ge style construction are semi-regular. The semi-regularity assumption essentially states that solving this system of polynomials is as hard as solving a random system of equations. If this were to the contrary, one might deduce that the degree of regularity bound provided by the Hilbert series is not tight, and hence a sharper bound could be found. This would imply that the complexity analysis could be improved and thus possibly might lead towards a classical algorithm for solving the* `PoSSoWN` *problem that is not exponential to the very least.*

5.3.2 Arora-Ge Method with Linearization (For $q = 2$ and $m = \mathcal{O}(n^2)$)

In this subsection we consider the particular sub-case of the $\texttt{PoSSoWN}_q$ when $q = 2$ and the number of samples available $m = \mathcal{O}(n^2)$. Given a system of ϵn^2 quadratic polynomials in n variables, we can use linearization techniques in conjunction with the above mentioned Arora-Ge technique to solve $\texttt{PoSSoWN}_2$. The product $P(\eta) = \eta(\eta - 1)$ produces an equation of degree 4. We have ϵn^2 such equations. Using

linearization, we can produce a system of ϵn^2 quadratic equations in $n^2/2$ variables $y_{ij} = x_i x_j$. Additionally, a monomial $x_a x_b x_c x_d$ be written in three possible manner.

$$(x_a x_b)(x_c x_d) = (x_a x_c)(x_b x_d) = (x_a x_d)(x_b x_c) = y_{ab} y_{cd} = y_{ac} y_{bd} = y_{ad} y_{bc},$$

There are $n^4/4!$ different ways to choose the 4-tuples of distinct indices, and each choice gives rise to 2 equations. We thus have about $n^4/12$ additional quadratic equations in $n^2/2$ y_{ij} variables. This number of variables can be reduced to about $(1/2-\epsilon)n^2$ by replacing each of y_{ij} variables by its parametric representation as a linear combination of new variables z_k. These $(\epsilon n^2 + n^4/12)$ quadratic equations in the new $(1/2-\epsilon)n^2$ can be linearized again by replacing each product $z_i z_j$ by a new variable v_{ij}. This new system has $(\epsilon n^2 + n^4/12)$ linear equations in $((1/2-\epsilon)n^2)^2/2$ variables v_{ij}. This new system is uniquely solvable when

$$(\epsilon n^2 + n^4/12) \geq ((1/2 - \epsilon)n^2)^2/2.$$

If this previous condition holds true, then the system of equations originating from the Arora-Ge method can be solved by linearization, however, in the other case, the previously proposed method from Section 5.3.1 of using Gröbner basis techniques still holds.

5.3.3 Exhaustive Search

We also consider the class of combinatorial attacks on the `PoSSoWN` problem. Such attacks can be mounted on either the secret directly, or could be used to recover the noise in order to recover the `PoSSo` instance from the `PoSSoWN` instance. We present the two algorithms as follows.

The first type of exhaustive search is over all the possible values of the secret $\mathbf{s} \in \mathbb{F}_q^n$. Since $\mathbf{s}$ is a vector with dimension n, hence in the worst-case scenario, the attacker has to compute q^n possible solutions. For an instance of `PoSSo`, evaluating the public-key polynomials over q^n candidate solutions and comparing with the ciphertexts is the method of pruning. However, for the case of `PoSSoWN`, without the knowledge of the error, such an attack is not possible. This can be used in conjunction with other attacks such as the previously mentioned Arora-Ge approach (see Section 5.3.1) that proposes a method of disregarding the error by constructing a higher degree polynomial. The complexity of such an attack depends on the degree of the polynomial P as defined in Equation (5.2). In particular, from the complexity results of enumerating the common zeros of a system of multivariate polynomials over $\mathbb{F}_2[x_1, \ldots, x_n]$ [BCC+10], the expected number of operations is $4(4k+2)\log_2 n \cdot 2^n$ where k is the upper bound on the magnitude of the error.

Another possible way to look at a system of m noisy quadratic equations over n variables is to consider it as a system of m equations over $n+m$ variables, i.e counting the errors as unknown variables. Thus enumerating the errors with exhaustive search, the problem reduces to just solving a system of m quadratic equations in n variables. Recall from Section 2.2, [LPT+17] introduces such a method that allows us to solve the system of equations. Once the error values have been recovered, substituting them back into the system of noisy polynomials yields an instance of PoSSo. Now [LPT+17] states the time complexity is

$$\mathcal{O}\left(q^n \cdot \left(\frac{\log q}{2e}\right)^{-n}\right), \quad \text{where } e = 2.718\ldots \text{the Napier constant,}$$

for finding the satisfiability of a system of equations where the solutions are in $\mathbb{F}_q$. So the total time complexity of performing an exhaustive search over the error and then the proposed algorithm of [LPT+17] is therefore

$$k^m \cdot \mathcal{O}\left(q^n \cdot \left(\frac{\log q}{2e}\right)^{-n}\right).$$

In this previous approach, instead of performing exhaustive search or a fast brute force attack of [LPT+17], one can use Gröbner basis to solve the system of equations. Using F5 [Fau02], the expected number of operations is upper bounded by

$$\mathcal{O}\left(m \cdot D_{reg}\left(\binom{n + D_{reg}}{D_{reg}}^{\omega}\right)\right).$$

Hence, the total complexity including the exhaustive search over the errors is

$$k^m \cdot \left(m \cdot D_{reg}\left(\binom{n + D_{reg}}{D_{reg}}^{\omega}\right)\right),$$

where D_{reg} is the degree of regularity over the system of m quadratic equations in n variables (Refer to Proposition 2.3.36 from Section 2.3.3).

5.3.4 Max-PoSSo Gröbner Basis Attack

In this section, we propose an algorithm which solves an instance of the Max-PoSSo problem (see Section 5.1), and whose solution can be turned into a solution for an instance PoSSoWN problem. Previously, we performed a Gröbner basis attack on the input system of equations which were erroneous. Without loss of generality, we can assume that only a certain fraction of the equations is non-noisy. So if we select only such non-noisy equations and then solve the Gröbner basis just over these non noisy equations gives us a solution (or a set of solutions) to the system of equations.

Suppose we are given a system of m equations in n variables. Let us assume

that only $t < m$ equations are error-free. Therefore, one can perform Gröbner basis computation over this subsystem of t quadratic equations over n variables, whose complexity is upper bounded by

$$\left(tD_{reg}\binom{n+D_{reg}}{D_{reg}}^{\omega}\right)$$

Since, we perform this for a particular correct choice of t equations, the number of expected operations for the entire attack is upper bounded by

$$\binom{m}{t} \cdot \left(tD_{reg}\binom{n+D_{reg}}{D_{reg}}^{\omega}\right)$$

Furthermore, one can also perform an exhaustive search for this fraction of t equations and then compute the Gröbner basis of the corresponding subsystem. This formalization of the problem differs from the attack presented in the previous Section 5.3.2. It should be noted that, the degree of regularity of the two systems in consideration vary since in the previous approach, the number of equations $m > n$, while now we consider only a fraction t of these m polynomials.

5.4 Conclusion

In this chapter, we provide the first complete state-of-the-art of the `PoSSoWN` problem, the noisy variant of the well known `NP`-Hard `PoSSo` problem. We show that the hardness of the `PoSSoWN` can be reduced to some well known hard problems. We also define the decision and the search variants of the problem and show that there is an equivalence between these variants when the number of samples is bounded by some polynomial factor of the number of variables. Additionally, we provide a survey of all known algorithms which solve the `PoSSoWN` problem which include some techniques to solve the `LWE` problem modified as algorithms for `PoSSoWN`. We conclude that the `PoSSoWN` problem is a good candidate to construct multivariate cryptosystems that are guaranteed post-quantum security via the hardness of the hard problem.

Chapter 6

CFPKM: A Submission to NIST

Abstract

The problem of solving a system of noisy polynomial equations has been shown to be a NP-Hard problem. As a submission to the NIST Post-quantum standardization completion, we designed a new multivariate key encapsulation scheme based on this hard problem. We provided a new design which takes use of errors in polynomials in order to blind the information passed over open channels. We describe the scheme along with security analysis and complete analysis of the potential algebraic attacks on it along with a proposal of two security parameters satisfying various security strengths. Finally we show why because of structural defect, the scheme was broken.

6.1 Background

In Chapter 3, we discussed Public-key cryptosystems in great detail. However, public-key algorithms are generally very slow as compared to symmetric cryptographic algorithms, one of the main reasons being a large key size resulting in a larger requirement of computation power. Therefore, it is not efficient to use public-key algorithms when Bob wants to send large amounts of data. But if public-key encryption techniques are used to exchange a shared key between Alice and Bob, by using a fast symmetric mechanism with the shared key, Bob can send large amounts of data to Alice much more efficiently.

However, as mentioned previously, symmetric cryptosystems suffer from an "issue". The parties need to exchange the secret-key before the cryptosystem is ready for use. A *key-exchange* mechanism provides a method to do this. A key-exchange scheme works as follows: two parties exchange a sequence of information, without exchanging the actual secret key. The key-exchange is also equipped with a reconciliation mechanism such that both users can agree to the same key.

Previously in Chapter 1, we also talked about another possible way a key-exchange can be done: via public-key algorithms. These types of public-key algorithms are also commonly known as *Key Encapsulation Mechanisms* (KEM). It is not hard to see that using such techniques, one can convert any encryption-decryption public-key cryptosystem into a key encapsulation scheme.

Recently many key-exchange schemes have been published, most famously by Ding *et. al* in [DXL12], Chris Peikert in [Pei14] and Costello *et al* [BCD+16] to name a few. Peikert in his paper [Pei14] detailed the construction of a passively secure KEM based on the key exchange using an innovative key reconciliation mechanism.

As already mentioned, the `NIST` PQC standardization process involved the call of proposals for post-quantum key exchange schemes. Amongst all the submissions, the majority of the key-exchange cryptosystems that have been proposed are based on the hard problem of LWE, which are lattice-based. In this chapter, we present a KEM, that is based on the hard problem of `PoSSoWN`, which we discussed in great detail in Chapter 5.

6.2 Passively Secure KEM

In this section we construct a KEM, based on problem of solving a system of polynomials with error, *i.e* `PoSSoWN`. First we describe the design an un-authenticated key exchange protocol.

6.2.1 Parameter Space

The scheme involves the following parameters :

1. q, a large positive prime power, which defines the finite field $\mathbb{F}_q$ for the Polynomial Ring $\mathbb{P}$, where we define our system of equations. It is taken of the form of 2^k for some $k \in \mathbb{Z}_+$,

2. $\mathbb{Z}_q$ defines the field of integers modulus q,

3. n, the number of variables which defines the Polynomial Ring $\mathbb{P}$,

4. m, number of equations in the system of equations,

5. s, is an integer which defines the range of values from where the secret and errors are chosen uniformly,

6. B, is the number of most significant bits which are chosen to create a session key,

6.2.2 Construction

6.2.2.1 Secret-key and Public-key

The secret-key is a concatenation of a random seed value and a secret vector $\mathbf{sa} \in [0, s]^n$ chosen randomly from a uniform distribution $\mathcal{U}_s^n$. The seed is used to generate a sequence of coefficients over $\mathbb{F}_q$ from the range $[0, q^\alpha]$ which are used to build a system of generic quadratic polynomials $f'_1, \ldots, f'_m \in \mathbb{F}_q[x_1, \ldots, x_n]$. The secret-key has the following structure,

$$\texttt{SK} = (\text{seed} \mid\mid \mathbf{sa}).$$

The public-key in CPFKM is a concatenation of the same *seed* value and a vector $\mathbf{b1} \in \mathbb{F}_q^m$. This vector $\mathbf{b1}$ is result of evaluating the set of quadratic polynomials $(f_1, \ldots, f_m) \in \mathbb{F}_q^m[x_1, \ldots, x_n]$ over the chosen secret vector $\mathbf{sa}$, where each f_i is given by the following

$$f_i(x_1, \ldots, x_n) := f'_i(x_1, \ldots, x_n) + e_i, \quad e_i \in \langle 0, s \rangle.$$

Therefore, each i^{th} component of the vector is defined as $\mathbf{b1}_i = f_i(\mathbf{sa})$. The public-key has the following structure

$$\texttt{PK} = (\text{seed} \mid\mid \mathbf{b1}).$$

6.2.2.2 The Algorithm

The Key Encapsulation is defined by three main algorithms namely `KeyGen`, `Encaps` and `Decaps`. The protocol has been has been summarized in Figure 6.1.

KeyPair Generation

Function : `Keygen`()

Code Function : *crpto_kem_keypair*(`PK`,`SK`).

The public-key and the secret-key are generated as a part of the `KeypairGen` function. The function generates a random value namely, *seed*. It makes use of another internal function called `PolGen` which, using the input of *seed*, generates a system of m multivariate quadratic polynomials $(f_1, \ldots, f_m) \in \mathbb{F}_q[x_1, \ldots, x_n]$. The *seed*, which is input to this function, is further input to the random function which pseudo-randomly generates the coefficients.

This `PolGen` function has been summarized below

- For each of the m quadratic polynomial f_i create a structure of three vectors $\texttt{QD} \in \mathbb{Z}_{q^\alpha}^{n^2}, \texttt{L} \in \mathbb{Z}_{q^\alpha}^{n}, \texttt{C} \in \mathbb{Z}_{q^\alpha}$. The structure holds the coefficients of the

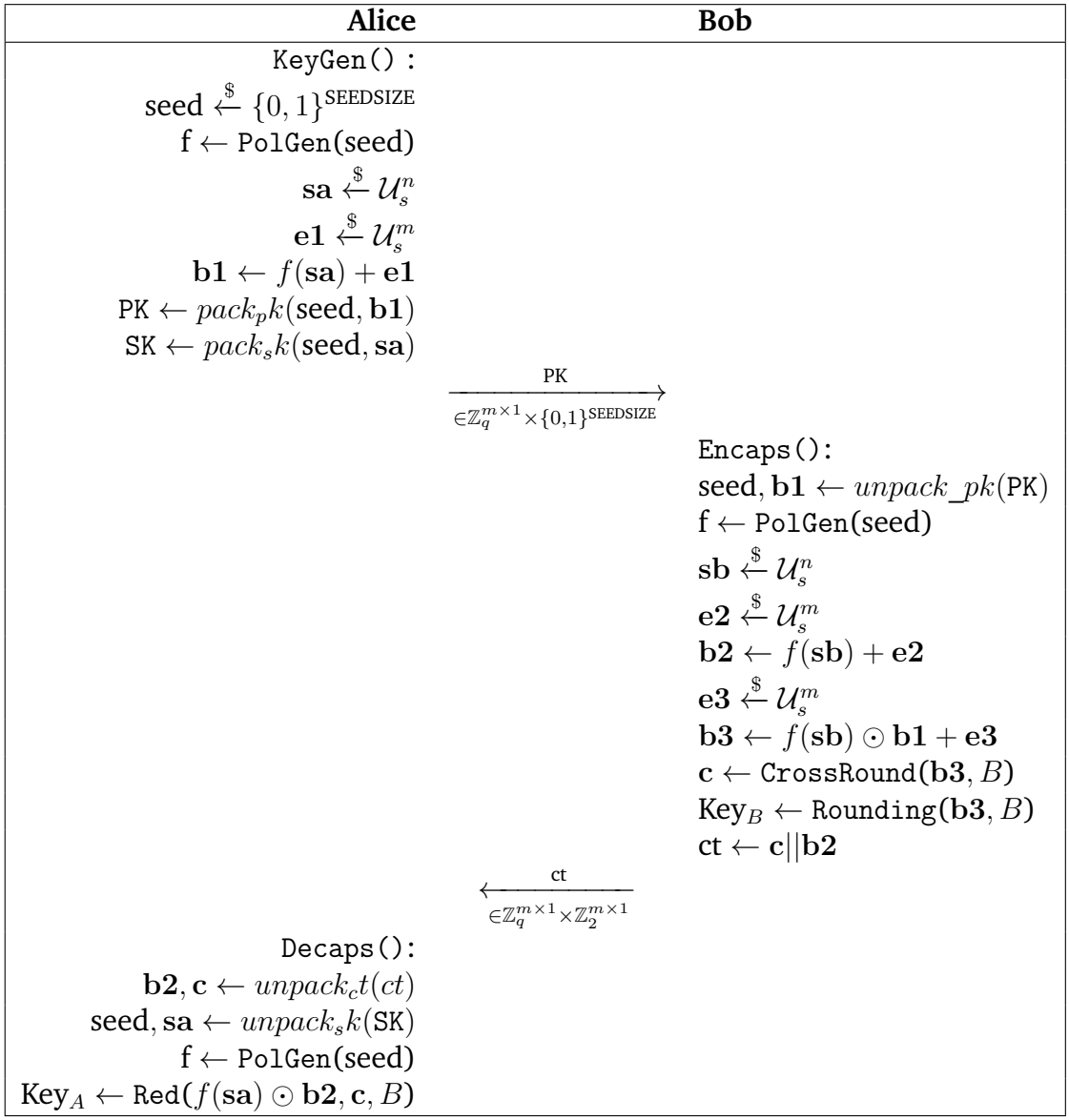

Figure 6.1 – Our KEM Scheme based on PoSSoWN

polynomials such that `QD` holds the coefficients of the quadratic monomials, `L` holds the coefficients for the linear monomials and `C` holds the constant term,

- using a random function and the *seed*, populate these vectors
- return f_i

Once the `PolGen` function creates the polynomials, the `KeyGen` algorithm randomly generates a secret vector **sa** of dimension n from the uniform distribution $\mathcal{U}_s^n$. An error vector $\mathbf{e1} \in \mathcal{U}_s^m$ is also generated. Each of the polynomials f_i, from the previously generated system of quadratic polynomials using `PolGen`, are evaluated over the secret vector **sa** and noise is added to them to generate another vector **b1** where

$$\mathbf{b1}_i = f_i(\mathbf{sa}) + \mathbf{e1}_i \mod q,$$

for i^{th} component of the vector.

Finally the Public Key `PK` is constructed by concatenating the *seed* along with this vector **b1** using the *pack_pk* function. The Secret Key `SK` is formed by concatenating *seed* and the secret vector **sa** using the *pack_sk* function. The function then outputs the `PK` and `SK`.

Key Encapsulation

Function : `Encaps`()

Code Function : *crypto_kem_enc*()

The encapsulation process encodes the shared secret using the public-key of Alice. It takes use of some extra functions which have been defined below.

1. `CrossRound`(w, B): This function takes in an integer $w \in [0, q)$ and given B, outputs the $(B+1)$'th most significant bit of $\log q$-bit binary representation of w, which has been referred to as the *crossround bit*.

   $$\texttt{CrossRound}(w, B) = \lfloor w \cdot 2^{-\bar{B}+1} \rfloor \mod 2.$$

   The function can also be extended to a vector of integers $\mathbf{w} \in [0, q)^m$. Thus on an input of a vector, `CrossRound` works independently on each component of vector and outputs another vector carrying the *crossround bit*.

2. `Rounding`(w, B): This function takes in an integer $w \in \mathbb{Z}_+$ and given B, outputs the B most significant bits of $\log q$-bit binary representation of $(w + 2^{\bar{B}-1}) \mod q$, where $\bar{B} = \lceil \log q \rceil - B$.

   $$\texttt{Rounding}(w, B) = \lfloor ((w + 2^{\bar{B}-1}) \mod q) \cdot 2^{-\bar{B}} \rfloor,$$

where $\lfloor \cdot \rfloor$ is the Floor function. Similar to the previous function, this can also be extended to a vector of integers $\mathbf{w} \in \mathbb{Z}_+^m$. Thus on an input of a vector, Rounding works independently on each component of vector and outputs another vector carrying the Rounding value of each component.

3. $\odot$: This function takes in two vectors $\mathbf{a}$ and $\mathbf{b}$ returns a vector $\mathbf{y}$ which is a component wise scalar product of $\mathbf{a}$ and $\mathbf{b}$. I.e., $\mathbf{y}_i = \mathbf{a}_i \cdot \mathbf{b}_i$ for each i^{th} component of the vectors.

Thus the function for Key encapsulation follows the following procedure for creation and encapsulation of the shared key.

1. `Encaps`() takes in the `PK` and then uses the *unpack_pk* process to get $\mathbf{b_1}$ and the *seed*.
2. Uses the *seed* and the `PolGen` function to generate the same system of quadratic polynomials $(f_1, \ldots, f_m) \in \mathbb{F}_q^m[x_1, \ldots, x_n]$.
3. Randomly sample vectors $\mathbf{sb} \leftarrow \mathcal{U}_s^n$, $\mathbf{e2} \leftarrow \mathcal{U}_s^m$ and $\mathbf{e3} \leftarrow \mathcal{U}_s^m$.
4. Computes $\mathbf{b2}_i = (f_i(\mathbf{sb}) + \mathbf{e2}_i) \mod q$ for each i^{th} component of the vector.
5. Compute $\mathbf{b3}_i = f_i(sb) \odot \mathbf{b1}_i + \mathbf{e3}_i$ for each of the i^{th} component.
6. Uses the `CrossRound`$(\mathbf{b3}, B)$ function over the vector $\mathbf{b3}$ to output a hint vector $\mathbf{c} \in \mathbb{Z}_2^m$.
7. The key for Bob, $\mathbf{Key_{Bob}}$ is derived using the `Rounding`$(\mathbf{b3}, B)$ function thus giving the B most significant bits of each of the i^{th} component of $\mathbf{b3}$.
8. Returns $ct = pack\_ct(\mathbf{b2}, \mathbf{c})$ and SS $= \mathbf{Key_{Bob}}$.

Key Decapsulation

Function : `Decaps`()

Code function : *crypto_kem_dec*(`SS`,ct,`SK`)

Alice does the decapsulation process, which uses the ciphertext *ct* from Bob and Alice's secret-key `SK`, to derive the shared secret-key `SS`. The *kem_dec* function calls another function called `Red`. The function is described below.

`Red`(w, c, B): On input of vectors $\mathbf{w} \in \mathbb{Z}_+^m$ and $\mathbf{c} \in \mathbb{Z}_2^m$, Red$(\mathbf{w}, c, B)$ outputs `Rounding`$(\mathbf{v}, B)$, where $\mathbf{v}$ is the closest element to $\mathbf{w}$ such that `CrossRound`$(\mathbf{v}, B)$ = c. This function takes in $\mathbf{w} = f(\mathbf{sa}) \odot \mathbf{b2}$, and follows the procedure below for each i^{th} component of the vector independently,

- checks if $\texttt{CrossRound}(\mathbf{w_i} \mod q, B)$ is equal to $\mathbf{c_i}$ or not. If its true, then it returns $\texttt{Rounding}(\mathbf{w_i}, B)$.
- If the value is false, then it adds $2^{\bar{B}-2}-1$ to $\mathbf{w_i}$ and then checks if $\texttt{CrossRound}(\mathbf{w_i}+2^{\bar{B}-2}-1 \mod q, B)$ is equal to $\mathbf{c_i}$ or not. If true then returns $\texttt{Rounding}(\mathbf{w_i}+2^{\bar{B}-2}-1, B)$
- If still false, then subtracts $2^{\bar{B}-2}-1$ from $\mathbf{w_i}$ and checks if $\texttt{CrossRound}(\mathbf{w_i}-2^{\bar{B}-2}+1 \mod q, B)$ is equal to $\mathbf{c_i}$ or not. If true then returns $\texttt{Rounding}(\mathbf{w_i}-2^{\bar{B}-2}+1, B)$.
- If still false, then it returns 0 .

Overall decapsulation function follows the steps below

1. Uses *unpack_sk*(SK) to get the secret vector **sa** and *seed* used by Alice earlier to generate her public-key PK.
2. Unpacks the ciphertext *ct* using *unpack_ct*, to get **b2** and the hint vector **c**
3. Use the *seed* to generate the same system of polynomials f_i.
4. Computes $f_i(\mathbf{sa}) \odot \mathbf{b2_i}$ for each ith component.
5. Calls the Red function on the input of $f(\mathbf{sa}) \odot \mathbf{b2}$, the hint vector **c** and B, the number of bits over which the key reconciliation has been agreed upon. The Red function outputs $\texttt{SS}= \mathbf{Key_{Alice}}$.
6. Returns $\mathbf{Key_{Alice}}$.

6.2.3 Correctness

We consider a large modulus q as a power of 2. For a choice of $1 \leq B < \log q - 1$ let, $\bar{B} = \log q - B$. We define the following terms just for purpose of our proof.

Definition 6.2.1 (Interval)**.** *A set of $2^{\bar{B}}$ consecutive positive integers which is represented as $[i \cdot 2^{\bar{B}}, (i+1) \cdot 2^{\bar{B}} - 1]$ for $i = 0$ to ∞.*

Definition 6.2.2 (Sub-interval)**.** *A set of $2^{\bar{B}-1}$ consecutive positive integers which is represented as $[i \cdot 2^{\bar{B}-1}, (i+1) \cdot 2^{\bar{B}-1} - 1]$ for $i = 0$ to ∞.*

Thus an interval has positive integers with the same most significant bits except the $\bar{B}$ least significant bits in their binary representation, whereas a sub-interval has positive integers with the same most significant bits except for the $\bar{B}-1$ least significant bits. Thus, a sub-interval splits up an interval equally according to their $\bar{B}$th least significant bit.

Let us denote a simple modulus map $h : \mathbb{Z}_+ \to [0, q-1]$ as

$$h(v) = v \mod q.$$

The idea of this modulus map can also be extended to a set of integers. Therefore, for any set of integers can be mapped to a set of integers in the range $[0, q]$ with the h map.

Lemma 6.2.3. *Suppose we have a large modulus q being a power of 2. With the definition of the modulus map as above, a sub-interval $I \in \mathbb{Z}_+$ maps to a sub-interval in $[0, q-1]$, i.e. $h(I) \in [0, q-1]$ is another sub-interval.*

Proof. Now we have $q = 2^k$, such that k is large enough, then $q \mod 2^{\bar{B}-1} = 0$. Thus it is a starting value/point of some sub-interval. So for any sub-interval $I = [i \cdot 2^{\bar{B}-1}, (i+1) \cdot 2^{\bar{B}-1} - 1]$,

$$h(I) = [i \cdot 2^{\bar{B}-1} \mod q, ((i+1) \cdot 2^{\bar{B}-1} - 1) \mod q]$$

$$= [i \cdot 2^{\bar{B}-1} \mod 2^k, ((i+1) \cdot 2^{\bar{B}-1} - 1) \mod 2^k]$$

$$= [i \cdot 2^{\bar{B}-1} \mod (2^{k-\bar{B}+1} \cdot 2^{\bar{B}-1}), ((i+1) \cdot 2^{\bar{B}-1} - 1) \mod (2^{k-\bar{B}+1} \cdot 2^{\bar{B}-1})]$$

$$= [(i \mod 2^{k-\bar{B}+1}) \cdot 2^{\bar{B}-1}, (((i+1) \mod 2^{k-\bar{B}+1}) \cdot 2^{\bar{B}-1} - 1) \mod 2^k]$$

$$= [(i \mod 2^{k-\bar{B}+1}) \cdot 2^{\bar{B}-1}, ((i \mod 2^{k-\bar{B}+1} + 1) \cdot 2^{\bar{B}-1} - 1) \mod 2^k]$$

$$= [j \cdot 2^{\bar{B}-1}, ((j+1) \cdot 2^{\bar{B}-1} - 1) \mod 2^k]$$

Now $j < 2^{k-\bar{B}+1}$, this implies that $(j+1) \cdot 2^{\bar{B}-1} \leq 2^k$, which means that $(j+1) \cdot 2^{\bar{B}-1} - 1 < 2^k$. Hence we can write

$$h(I) = [j \cdot 2^{\bar{B}-1}, (j+1) \cdot 2^{\bar{B}-1} - 1],$$

where $j = (i \ mod \ 2^{k-\bar{B}+1})$ is an integer. We see that $h(I)$ is also has a form of an sub-interval. Thus any sub-interval $\in \mathbb{Z}_+$ is mapped to some sub-interval $I' = h(I) \subset [0, q-1]$. □

Assumption 6.2.4. *We assume that any integer in $[0, q-1]$ has a binary representation in $\log q$ bits.*

Lemma 6.2.5. *For a large modulus $q = 2^k$, when two positive integers v and w lie in the same sub-interval, their crossround bits are same, while when in adjacent intervals, then their crossround bits are different.*

Proof. Let's assume they lie in the same sub-interval I, then

$$v, w \in I = [i \cdot 2^{\bar{B}-1}, (i+1) \cdot 2^{\bar{B}-1} - 1]$$

for some particular value of i. From the definition of the mapping h, we see that $v \mod q \in h(I)$ and $w \mod q \in h(I)$, i.e. they are in the same sub-interval $h(I)$. This implies that the $\bar{B}$th least significant bit of $v \mod q$ and $w \mod q$ are the same, since in an sub-interval all the bits except the $\bar{B}-1$ least significant bits are same for all integers in the sub-interval (Definition 6.2.2). And from the definition of the `CrossRound` function, this $\bar{B}$th least significant bit is the *crossround* bit. Hence when both v and w are in the same sub-interval, their *crossround* bits are equal.

Now let v and w lie in two adjacent sub-intervals. We denote the two sub-intervals as $v \in I_1 = [i \cdot 2^{\bar{B}-1}, (i+1) \cdot 2^{\bar{B}-1} - 1]$ and $w \in I_2 = [(i+1) \cdot 2^{\bar{B}-1}, (i+2) \cdot 2^{\bar{B}-1} - 1]$ for some i. Then I_1 maps onto some sub-interval $I_1' = h(I_1) \subset [0, q-1]$ and I_2 onto $I_2' = h(I_2) \subset [0, q-1]$. Here

$$I_1' = h(I_1) = [j \cdot 2^{\bar{B}-1}, (j+1) \cdot 2^{\bar{B}-1} - 1]$$

$$I_2' = h(I_2) = [((j+1) \mod 2^{k-\bar{B}+1}) \cdot 2^{\bar{B}-1}, ((j+2) \mod 2^{k-\bar{B}+1}) \cdot 2^{\bar{B}-1} - 1]$$

where $j = (i \mod 2^{k-\bar{B}+1})$. As $j \neq ((j+1) \mod 2^{k-\bar{B}+1})$, hence $I_1' \neq I_2'$, i.e. they do not map on to the same sub-interval in $[0, q-1]$.
It is also important to note that as h is just a simple modulus map and the modulus q is a power of 2, so for any $v \in \mathbb{Z}_+$, $h(v)$ is the $\log q$ least significant bits of binary representation of v. Hence there is no change in the $\bar{B}$th least significant bit after the modulo operation.

As I_1 and I_2 are two adjacent sub-intervals, they differ by their $\bar{B}$th least significant bit. This implies that $h(I_1)$ and $h(I_2)$ have different $\bar{B}$th least significant bit, as the mapping does not change the $\log q$ least significant bits of any integer in I_1 or I_2.
Now $v \mod q \in h(I_1)$ and $w \mod q \in h(I_2)$. So their *crossround* bits, which is the $\bar{B}$th least significant bit are different. □

Thus now to prove the correctness of the key reconciliation algorithm, we propose the following Theorem.

Theorem 6.2.6. *Let the choice of a large modulus q which is a power of 2. Let us represent for any ith component of the vectors $\mathbf{b3}$ and $f(\mathbf{sa}) \cdot \mathbf{b2}$ as $v = \mathbf{b3}_i \in \mathbb{Z}_+$ and $w = f_i(\mathbf{sa}) \cdot \mathbf{b2}_i \in \mathbb{Z}_+$. The above Key exchange protocol is correct with a high probability i.e. $\mathbf{K_{Alice}} = \mathbf{K_{Bob}}$ when for all i components of the vectors of dimension m,*

$$w \in (v' - 2^{\bar{B}-2} - 1, v] \cup [v, v'' + 2^{\bar{B}-2} + 1)$$

where $v' = \lfloor v \cdot 2^{-\bar{B}+1} \rfloor \cdot 2^{\bar{B}-1}$, $v'' = v' + 2^{\bar{B}-1}$ *and* $\bar{B} = \lceil \log q \rceil - B$. B *is the number of most significant bits chosen for key agreement such that* $1 \leq B < \lceil \log q \rceil - 1$ *and* $|\cdot|$ *gives the absolute value component wise.*

Proof. Let us consider the integers, $\mathbf{b3}_1 = v \in \mathbb{Z}_+$ and $f_1(\mathbf{sa}) \cdot \mathbf{b2}_1 = w \in \mathbb{Z}_+$. Consider the sub-intervals I_1 and I_2 such that $v \in I_1$ and $w \in I_2$.

1. **Case 1** : When v and w lie in the same sub-interval,i.e. $I_1 = I_2$. So from Lemma 6.2.5, we conclude that `CrossRound`(v mod q,B) = `CrossRound`(w mod q,B).
   Thus Alice performs `Rounding`(w, B) as a part of the `Red` function to get $\lfloor ((w + 2^{\bar{B}-1}) \mod q) \cdot 2^{-\bar{B}} \rfloor$ and Bob does `Rounding`(v, B)= $\lfloor ((v + 2^{\bar{B}-1}) \mod q) \cdot 2^{-\bar{B}} \rfloor$.
   Since both v and w are in same sub-interval, this implies that $v + 2^{\bar{B}-1}$ and $w + 2^{\bar{B}-1}$ are also in same sub-interval. Now from Lemma 6.2.3,we can infer that both $(v + 2^{\bar{B}-1} \mod q)$ and $(w + 2^{\bar{B}-1} \mod q)$ lie again in the same sub-interval in $[0, q-1]$. We assumed that any integer in $[0, q-1]$ has a $\log q$-bit binary representation, Hence the B most significant bits for both are also equal, as in a sub-interval for any two integers all the bits except the $\bar{B} - 1$ least significant bits are equal. Hence we conclude that

$$\lfloor ((w + 2^{\bar{B}-1}) \mod q) \cdot 2^{-\bar{B}} \rfloor = \lfloor ((v + 2^{\bar{B}-1}) \mod q) \cdot 2^{-\bar{B}} \rfloor$$

$$\implies \mathbf{K_{Alice}} = \mathbf{K_{Bob}}$$

2. **Case 2**: When v and w lie in adjacent intervals. From Lemma 6.2.5, `CrossRound`(v mod q,B) $\neq$ `CrossRound`(w mod q,B).
   So, according to the Red function, first we add $2^{\bar{B}-2} - 1$ to w to get $w' = w + 2^{\bar{B}-2} - 1$. Now, as per Lemma 6.2.7, only two sub-cases are possible, either $w' \in I_1$ or $w' \in I_2$.

   If $w' \in I_1$, then this implies that by Lemma 6.2.5, `CrossRound`(v mod q,B) = `CrossRound`(w' mod q,B). This is another instance of Case 1. Therefore Alice performs `Rounding`(w', B) inside the `Red` function and Bob computes `Rounding`(v, B). As per Case 1 we conclude, $\mathbf{K_{Alice}} = \mathbf{K_{Bob}}$.

   Now consider the other sub-case, i.e. when $w' \in I_2$. By Lemma 6.2.5 it means that `CrossRound`(v mod q,B) $\neq$ `CrossRound`(w' mod q,B). Hence following steps of the `Red` function, we subtract, $2^{\bar{B}-2} - 1$ from w to get $w'' = w - 2^{\bar{B}-2} + 1$. Now again two sub-cases are possible (Lemma 6.2.7), $w'' \in I_1$ or $w'' \in I_2$.

   If $w'' \in I_1$, then we have `CrossRound`(v mod q,B) = `CrossRound`(w'' mod q,B) by Lemma 6.2.5. We thus find another instance of Case 1. Hence,

Alice does $\texttt{Rounding}(w'', B)$ and Bob does $\texttt{Rounding}(v, B)$, which gives us $\mathbf{K_{Alice}} = \mathbf{K_{Bob}}$.

Now let us look at the remaining case of $w'' \in I_2$. At this stage of Red function, we already have that $w' \in I_2$ and $w \in I_2$ while $v \in I_1$ and $I_1 \neq I_2$. $w' \in I_2$ and $w'' \in I_2$ implies that w lies in middle of the sub-interval I_2. This means that

$$w \notin (v' - 2^{\bar{B}-2} - 1, v' - 1] \cup I_1 \cup [v'' + 1, v'' + 2^{\bar{B}-2} + 1)$$

But this is in contradiction to our initial assumption.

3. **Case 3**: When v and w lie in two different sub-intervals separated by at least one interval. This implies that $|w - v'| > 2^{\bar{B}-1}$ or $|w - v''| > 2^{\bar{B}-1}$. From our assumption, we have that if $w \notin I_1$, then either $|w - v'| < 2^{\bar{B}-2} - 1$ or $|w - v''| < 2^{\bar{B}-2} - 1$. We find a clear contradiction to our assumption.

So, if the assumption of our theorem holds, after the Rounding operation, the protocol produces the same key for Alice and Bob. □

Lemma 6.2.7. *Suppose we have two sub-intervals I_1 and I_2 such that for two positive integers, $v \in I_1$ and $w \in I_2$. We also have*

$$w \in (v' - 2^{\bar{B}-2} - 1, v' - 1] \cup I_1 \cup [v'' + 1, v'' + 2^{\bar{B}-2} + 1)$$

where $v' = \lfloor v \cdot 2^{-\bar{B}+1} \rfloor \cdot 2^{\bar{B}-1}$ and $v'' = v' + 2^{\bar{B}-1}$. For $w' = w + 2^{\bar{B}-2} - 1$ and $w'' = w - 2^{\bar{B}-2} + 1$, only two cases are possible, w' and w'' are either in I_1 or I_2.

Proof. To see that this is true, suppose $w' \notin I_1$ and $w' \notin I_2$. This means that $|w' - v''| > 2^{\bar{B}-1}$. Which further implies that $|w - v''| > 2^{\bar{B}-2} - 1$, which is in clear contradiction to our initial assumption about w being in the range given. So w' must lie in either I_1 or I_2. The proof is similar for w''. □

Now the assumption in the Theorem 6.2.6 is dependent on the range s. We would like to determine the range by fixing the rest of the parameters. Let the choice of q be 2^k. The choice of the range from which the error and secret is chosen, let's suppose be $s = Round(2^\beta)$.
From Theorem 6.2.6, we get

$$|\mathbf{b3}_i - f_i(\mathbf{sa}) \cdot \mathbf{b2}_i| < 2^{(\log q - B - 2)}$$

$$\implies |\mathbf{e_3}_i + \mathbf{e1}_i \cdot \mathbf{b2}_i - \mathbf{e2}_i \cdot \mathbf{b1}_i| < 2^{(\log q - B - 2)}$$

$$\implies s + s \cdot |\mathbf{b1}|_{max} + s \cdot |\mathbf{b2}|_{max} < 2^{(\log q - B - 2)}$$

If we replace $|\mathbf{b1}|_{max}$ and $|\mathbf{b2}|_{max}$ by $|b| = max(|\mathbf{b1}|_{max}, |\mathbf{b2}|_{max})$ we get

$$s + 2 \cdot s \cdot |b| < 2^{(\log q - B - 2)} \tag{6.1}$$

The choice of the coefficients is from $[0, q^\alpha]$, hence the maximum possible value of a coefficient is $q^\alpha \approx 2^{\alpha k}$. First we need to determine the maximum possible value for $|b|$.
Now,

$$b = f(s) + e$$

$$b = (\sum_{i,j}^{n} a_{ij} x_i x_j + \sum_{i}^{n} b_i x_i + c) + e$$

$$= \mathcal{O}(n^2) \cdot 2^{\alpha k} \cdot 2^{2\beta} + n \cdot 2^{\alpha k} \cdot 2^{\beta} + 2^{\alpha k} + 2^{\beta}$$

Now that we have a maximum value of $|b|$ in terms of n, we can now look at left-hand-side of Equation 6.1,

$$s + 2 \cdot s \cdot |b|$$

Replacing the corresponding values we obtain

$$LHS = 2^{3\beta} 2^{2\log n + \alpha k + 1} + 2^{2\beta}(2^{\log n \alpha k + 1} + 1) + 2^{\beta}(2^{\alpha k + 1} + 1) \tag{6.2}$$

Right-hand-side of Equation 6.1 is $2^{\log q - B - 2}$. Replacing we get

$$RHS = \frac{2^k}{2^{B+2}}$$

So putting together the LHS and RHS of Eq 6.1, we have

$$2^{3\beta} 2^{2\log n + \alpha k + 1} + 2^{2\beta}(2^{\log n \alpha k + 1} + 1) + 2^{\beta}(2^{\alpha k + 1} + 1) < \frac{2^k}{2^{B+2}} \tag{6.3}$$

Corollary 6.2.8 (Asymptotic Result)**.** *With the choice of large modulus $q = 2^k$, the above proposed protocol succeeds with high probability , if following holds*

$$\beta < \left(\frac{k(1-\alpha) - B - 2 - 2\log n}{3} \right)$$

Corollary 6.2.8 gives an approximate upper bound for the range $s = 2^\beta$. Thus choosing the range accordingly (see Figure 6.2) for sampling our error and secret, our key exchange and agreement works correctly with very high probability. Although for accurate working of the protocol with probability 1, we need a much stricter bound which can be obtained by solving the Equation 6.3 for the variable β, in terms of the other variables α, B and k. Through experiments, we find that a good choice of the other parameters are $k > 7 \log n$, $B = 25$ and $\alpha = 0.3$

The choice for B, the number of most significant bits chosen per sample has been set to 4 (for our recommended parameters), as compared to the reconciliation mechanism by Peikert [Pei14] which extracts a single bit per sample. This choice is backed by the fact that an exhaustive search for directly finding the shared secret will take at least 2^{4m} operations and our goal is that $4m$ should be larger than at least our initial target of 128 bits of classical security.

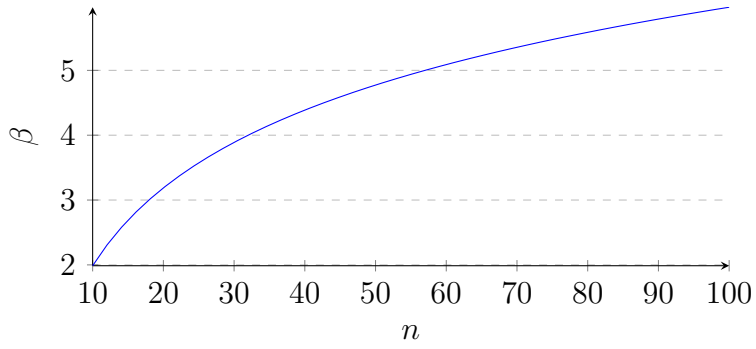

Figure 6.2 – Plot showing the relationship of β, the factor influencing the range s, with n for a choice of $k = 8 \log n$, $B = 4$ and $\alpha = 0.3$

6.2.4 Failure Rate

The assumption on which our key agreement works is that $f_i(\mathbf{sa}) \cdot \mathbf{b2}_i$ lies in a range of integers $(v' - 2^{\bar{B}-2} - 1, v] \cup [v, v'' + 2^{\bar{B}-2} + 1)$, where $v = \mathbf{b3}_i$, $v' = \lfloor v \cdot 2^{-\bar{B}+1} \rfloor \cdot 2^{\bar{B}-1}$ and $v'' = v' + 2^{\bar{B}-1}$. When the difference $|\mathbf{b3}_i - f_i(\mathbf{sa}) \cdot \mathbf{b2}_i|$ is less than $< q/2^{B+2}$, then it always works with probability 1. When the difference is $\geq 3q/2^{B+2}$ then the probability is 0. In between the two extremes of $2^{\bar{B}-2}$ and $3 \cdot 2^{\bar{B}-2}$, the probability of success decreases linearly. Let us denote the probability of success for a component of the m dimensional system as p_s. So

$$p_s = \begin{cases} 1, & \text{if } |v - w| < 2^{\bar{B}-2} \\ 0, & \text{if } |v - w| \geq 3 \cdot 2^{\bar{B}-2} \\ \frac{3 \cdot 2^{\bar{B}-2} - |v-w|}{2^{\bar{B}-1}} & \text{if } 2^{\bar{B}-2} \leq |v - w| < 3 \cdot 2^{\bar{B}-2} \end{cases} \tag{6.4}$$

The probability distribution of the distance $|v - w|$ is uniform, since all the parameters effecting the distance, namely, the error terms $e1, e2, e3$ and coefficients of f_i and the secrets **sa** and **sb** all follow an uniform distribution. So, from Equation 6.2, we can see that the error is bounded by

$$\text{Maxerr}(\beta) = 2^{3\beta} 2^{2 \log n + \alpha k + 1} + 2^{2\beta}(2^{\log n \alpha k + 1} + 1) + 2^{\beta}(2^{\alpha k + 1} + 1)$$

So, the total probability of success of the scheme is given by,

$$P_s(\beta) = \begin{cases} \left(\sum_0^a \frac{1}{\text{Maxerr}(\beta)} \right)^m = 1, & \text{if Maxerr}(\beta) < 2^{\bar{B}-2} \\ \left(\sum_0^a \frac{1}{\text{Maxerr}(\beta)} + \sum_{i=a+1}^b \frac{3 \cdot 2^{\bar{B}-2} - i}{2^{\bar{B}-1} \cdot \text{Maxerr}(\beta)} \right)^m, & \text{otherwise.} \end{cases} \tag{6.5}$$

where $a = min(2^{\bar{B}-2} - 1, \text{Maxerr}(\beta))$, $b = min(3 \cdot 2^{\bar{B}-2} - 1, \text{Maxerr}(\beta))$ and $m = n + 1$ is the number of samples that we use for our system/scheme. Thus the probability of failure of the scheme is given by $P_f(\beta) = 1 - P_s(\beta)$.

6.3 Analysis of Attacks Considered in Submission

In this section we present a summary of the main algebraic attacks against CFPKM. In Section 6.3.1 we consider the Arora-Ge method of solving a system of noisy equations by removing the error and then using Gröbner basis techniques. In Section 5.3.3, we consider the possibility of an exhaustive search over the secret. We also consider the exhaustive search over the secret and then using Gröbner basis techniques to solve the resultant system of equations with Gröbner basis.

In order to break our protocol, the main goal is to recover the secret **sa** or **sb** used by either of Alice or Bob. We have a system of equations

$$b1_1 = f_1(x) + e1_1$$

$$b1_2 = f_2(x) + e1_2$$

$$\vdots$$

$$b1_m = f_m(x) + e1_m$$

This is a system of non-linear multivariate polynomials, whose solution is $x = sa$. The process of finding this solution is the exact hard problem of PoSSoWN. In the following sections, we discuss the possible methods of solving this problem and thus provide the hardness results, which relates to our problem.

6.3.1 Arora-Ge Gröbner Basis Method

CFPKM requires solving a two systems of equations given by $\mathbf{b1} = f(\mathbf{sa}) + \mathbf{e1}$ and $\mathbf{b2} = f(\mathbf{sb}) + \mathbf{e2}$ where $f \in \mathbb{F}_q^m[x_1, \ldots, x_n]$ and $\mathbf{b1}, \mathbf{b2}, \mathbf{e1}, \mathbf{e2} \in \mathbb{F}_q^m$. The errors $\mathbf{e1}, \mathbf{e2}$ are chosen from a discrete uniform distribution from a range $[0, s]$.

We can rewrite the above polynomial equations as $\mathbf{e1} = \mathbf{b1} - f(\mathbf{sa})$ and $\mathbf{e2} = \mathbf{b2} - f(\mathbf{sb})$. In Section 5.3.1, we presented the approach of using Gröbner basis with the Arora-Ge approach. For our scheme, we can similarly construct the Arora-Ge polynomial (Equation (5.2)) as follows:

$$P(\eta) = \eta \prod_{j=1}^{s} (\eta - j),$$

where η is the error polynomial (See Section 5.3.1). Therefore, for each i^{th} component of the error vectors $\mathbf{e1} \in \mathbb{F}_q^m$ and $\mathbf{e2} \in \mathbb{F}_q^m$, we can substitute the error polynomials by $\mathbf{b1}_i - f_i(\mathbf{x})$ and $\mathbf{b2}_i - f_i(\mathbf{x})$ respectively where $\mathbf{x} = (x_1, \ldots, x_n)$.

Consequently, we get two system of polynomial equations corresponding to each of **e1** and **e2** respectively, and whose i^{th} components are given by,

$$P_i(\mathbf{b1}_i - f_i(\mathbf{x})) = (\mathbf{b1}_i - f_i(\mathbf{x})) \prod_{j=1}^{s} (\mathbf{b1}_i - f_i(\mathbf{x}) - j), \tag{6.6}$$

$$P_i(\mathbf{b2}_i - f_i(\mathbf{x})) = (\mathbf{b2}_i - f_i(\mathbf{x})) \prod_{j=1}^{s} (\mathbf{b2}_i - f_i(\mathbf{x}) - j). \tag{6.7}$$

We have two systems of m polynomials in n variables of degree $d = 2s + 2$, keeping in account of the degree of each f_i being 2 (i.e quadratic). It is quite intuitive to see that the polynomial P_i from Equation (6.6) equals zero when $\mathbf{x} = \mathbf{sa}$, and the same for Equation (6.7) when $\mathbf{x} = \mathbf{sb}$.
Additional constraints on the set of variables $(x_1, \ldots, x_n)$ come from the fact that even the secrets **sa** and **sb** are chosen from the range $[0, s]^n$.

$$x_1(x_1 - 1) \cdots (x_1 - s) = 0$$

$$x_2(x_2 - 1) \cdots (x_2 - s) = 0$$

$$\vdots$$

$$x_n(x_n - 1) \cdots (x_n - s) = 0$$

So if we are able to find a Gröbner basis of the system of equations along with system $P_i(\mathbf{x}) = 0$ of Equation (6.6) (or Equation (6.7)), then we will be able to recover **sa** (or **sb** respectively). The minimum number of samples that is assumed the attacker can have and running the F5 is $m' = n + 1$. In Table 6.1 we present the degree of regularities and the bit-complexities of the Arora-Ge Gröbner basis attack with the choice of $m' = n + 1$. The degree of regularity for our system is given by the degree of regularity for a semi-regular system of same parameters. Table 6.2 shows the time complexity of the Arora-Ge Gröbner basis attack on the `KEM` when different number of equations are made available to the attacker.

6.3.2 Exhaustive Search

In this section, we take use of the combinatorial attacks from Section 5.3.3.

6.3.2.1 Over the Shared Secret

An adversary can perform an exhaustive search over the shared secret constructed between Alice and Bob. The size of the shared key is determined by the total number of bits chosen for each component of the m-dimensional shared secret. Each component of the shared secret is of size B bits. Thus for an attacker to mount a

n	D_{reg}	Arora-Ge-GB($\omega := 2.35$)	Arora-Ge-GB($\omega := 2$)
10	11	51	44
15	14	70	60
20	17	88	77
25	19	104	90
30	22	122	106
35	25	141	121
40	42	195	169
45	46	217	186
50	50	238	204
55	55	262	224
60	59	283	243
65	63	304	261
70	67	325	279
75	71	346	297
80	75	367	315
85	79	389	333

Table 6.1 – Complexity of Arora-Ge Gröbner basis algorithm with $s \approx n^{0.25}$ and $m = n + 1$.

n	$m = (n(1 + 1/\log n))$	$m = n(\log n)^{(1/\epsilon)}$	$m = \mathcal{O}(n \log \log n)$
10	61	-	57
15	88	-	79
20	115	128	98
25	143	158	119
30	168	189	138
35	194	219	156
40	252	284	208
50	314	343	253
60	375	411	297

Table 6.2 – Complexity of Arora-Ge Gröbner basis algorithm with different m' other than $n + 1$ with $\omega = 2$.

brute force attack would require a time complexity of 2^{Bm} bit operations. Since, in our scheme, B has been set to 25 for both the set of proposed challenge parameters and m is at least greater than 4, we thus do not consider this attack anymore as the complexity of the attack is much higher than other algebraic attacks.

6.3.2.2 Over the Secret

Next we consider the brute force attack by an adversary to recover the hidden secrets $\mathbf{sa}, \mathbf{sb} \in \mathbb{F}_q^n$, as mentioned in Section 5.3.3. The secrets are chosen from $[0, s]^n$. Therefore, the attacker has to compute s^n possible solutions. Also, we assume that the range $s \approx n^\beta$, hence the number of operations in exhaustive search turns out to be $2n^{\beta n}$. Once these secrets have been recovered, with high probability the shared secret can be recovered by taking the B most significant bits of the product $f_i(\mathbf{sa}) \cdot f_i(\mathbf{sb})$.

Note 6.3.1. *If we increase the value of our range s, then the time for exhaustive search increases. But this in turn also increases the degree of regularity and hence also the time complexity of the Arora-Ge GB attack.*

6.3.2.3 Over the Error

Now, we take the other possibility of performing an exhaustive search over the errors. We take use of the technique presented in Section 5.3.3, where we consider the same system of noisy polynomials as a system of m equations over $n + m$ variables, i.e counting the errors as unknown variables. So once the error values have been recovered, for example $\mathbf{e1}$, we have the following system of equations

$$f_1(\mathbf{x}) - \mathbf{b1}'_1 = 0,$$

$$f_2(\mathbf{x}) - \mathbf{b1}'_2 = 0,$$

$$\vdots$$

$$f_m(\mathbf{x}) - \mathbf{b1}'_m = 0,$$

where $f_i(\mathbf{x}) \in \mathbb{F}_q[\mathbf{x}]$ with $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{b1}' = \mathbf{b1} - \mathbf{e1} \in \mathbb{F}_q^m$. We can solve this above system using the algorithm of [LPT+17] which has a complexity of

$$\mathcal{O}\left(s^n \cdot \left(\frac{\log s}{2e}\right)^{-n}\right),$$

where the values of x_i are chosen from the range $[0, s]$. Therefore, the total complexity of doing a exhaustive search of the error followed by the proposed algorithm of [LPT+17] is expected to be

$$s^m \cdot \mathcal{O}\left(s^n \cdot \left(\frac{\log s}{2e}\right)^{-n}\right).$$

6.3.3 Hybrid Attacks

In this section, we present the hybrid attacks which combines combinatorial methods, like exhaustive search, with algebraic techniques such as Gröbner basis methods. The first algorithm is the hybrid method of exhaustive search over the errors and Gröbner basis algorithms, which was presented in Section 5.3.3. For our scheme, the total complexity of this attack is

$$s^m \cdot \left(m \cdot D_{reg} \left(\binom{n + D_{reg}}{D_{reg}}^{\omega} \right) \right),$$

where D_{reg} is the degree of regularity over the system of m quadratic equations in n variables (Refer to Equation 5.3 for how to determine the D_{reg} in Section 5.3.1).

In Table 6.3 , we observe that an exhaustive search over the secret or the error is a faster and much efficient attack on our scheme. This is because our search space for both the secret as well as the error is much smaller than the finite field related to the polynomial ring in which the polynomials have been defined.

The next algorithm we present is the Max-PoSSo Gröbner basis attack that was presented in Section 5.3.4. Suppose we are given a system of m equations in n variables. Now the errors can be achieved by doing an exhaustive search. So now assuming that the error distribution is uniform over the range $[0, s]$, we can say that $t = m/(s+1)$ number of equations may be the exact solutions. That t equations are such that

$$f_1 = \cdots = f_t = 0,$$

where $f_1, \ldots, f_t \in \mathbb{F}_q[x_1, \ldots, x_n]$. Now the Gröbner basis attack on this sub-system takes

$$\left(tD_{reg} \binom{n + D_{reg}}{D_{reg}}^{\omega} \right),$$

number of operations, where D_{reg} is the degree of regularity of this sub-system. So, the total complexity of this attack turns out to be

$$\binom{m}{t} \cdot \left(tD_{reg} \binom{n + D_{reg}}{D_{reg}}^{\omega} \right).$$

Though comparing the performance of this attack with exhaustive search, we see that for $n = 20$, we are getting a security of 130 bits. But on the other hand, with $n = 30$ and the secret vector being chosen from a small range $[0, s]$, the exhaustive search performs much better. It is also important to mention that in case of this attack, the minimum number of equations m will not suffice with $n + 1$. It has to be much larger than this, big enough such that $t > n$ for the Gröbner basis attack to work. Hence we just report the performance of this attack in Table 6.3.

n	s	AG-GB	Ex-Sec	Ex-Err-SODA	Ex-Err-GB (D_{reg})	HYB
30	2	106	30	74	94 (9)	191
35	2	121	35	86	107 (10)	224
40	3	168	63	120	157 (14)	251
45	3	186	71	134	174 (15)	276
50	3	204	79	149	190 (16)	307
55	3	224	87	164	206 (17)	338
60	3	243	95	178	222 (18)	364
65	3	261	103	193	238 (19)	394
70	3	279	111	208	258 (21)	420
75	3	297	119	223	274 (22)	451
80	3	315	**129**	237	290 (23)	481
85	3	333	135	252	306 (24)	507
90	3	351	142	267	322 (25)	538
95	3	370	151	281	338 (26)	568
100	3	389	158	296	354 (27)	594
105	3	404	166	311	374 (29)	624
110	3	425	174	325	390 (30)	649
115	3	442	**182**	340	406 (31)	680
120	3	460	190	355	422 (32)	711

Table 6.3 – Comparing time complexity with $s \approx n^{0.25}$, all time complexity values in $\log_2$. The column AG-GB represents the Arora-Ge style Gröbner basis attack, EX-Sec represents Exhaustive search over the Secret sa, EX-Err represents the Exhaustive search over the errors and then using SODA and Gröbner basis algorithms and finally HYB represents the hybrid approach from Section 6.3.3

6.4 Detailed Performance Analysis

The scheme is written in `C++` language. In this following table we list the description of the platform on which the scheme has been developed and tested for correctness.

Computer	OS	Architecture	Processor	Frequency
Laptop	Linux Mint 18.1	x86_64	i7-6600U	2.60 GHz

RAM	Version of gcc
31.3 Gb	gcc 5.4.0

Table 6.4 – Platform for designing `CFPKM`

6.4.1 Time

The following measurements are for the KEM. For the measures, it runs a number of tests such that the global used time is greater than 10 seconds and the global time is divided by the number of tests. For our scheme with `CFPKM128` the key generation, takes 72 ms. The key encapsulation scheme takes on an average about 108 ms (over a run of 30 tests). The decapsulation of the shared secret key takes about 143 ms. For `CFPKM182` key generation takes 120 ms. The key-encapsulation takes for this parameter 150 ms on an average of 30 tests, while the decapsulation takes about 190 ms.

6.4.2 Space

The key sizes of the parameters are calculated directly (and confirmed in various experiments) from the structure of keys. Recall that the public key was constructed by concatenating the *seed* value and the public vector **b1**, while the secret key is a concatenation of the same *seed* value and the secret **sa**. The ciphertext comprises of two vectors, the crossround bit vector **c** and the vector **b1**.

The total size of public key for `CFPKM128` turns out to be 696 bytes. The secret key is 128 bytes for the parameters of `CFPKM128`. The ciphertexts, which are sent from Bob to Alice, are 729 bytes long, whereas the shared secret is of 81 bytes.

For `CFPKM182`, the public key size turns out to be 995 bytes which include a 48 byte long *seed* value. The secret key is 182 bytes long while the shared secret at the end of the key exchange is 116 bytes. The size of the ciphertexts from Bob to Alice amounts to 1044 bytes.

6.4.3 How parameters affect performance

The Key Encapsulation is mainly affected by the number of equations m in our system, the range s and also by the number of most significant bits B that we use in our scheme. From Section 6.2.4, we see that the scheme's failure probability is a function of the range s. This probability is over the m equations used in the scheme. Hence, the efficiency of the scheme is dependent on both s and m. Also from Section 6.3, the fastest attack, exhaustive search over the secret, tells us that the security of the scheme is a factor of both s and m.

6.5 Advantages and Limitations

`CFPKM`, is dependent on small secrets and errors, which is one limitation of the proposed scheme. It has been left a future work for further improving the performance of the scheme. But on the other hand, CFPKM has a lot of advantages.

The hardness of our key exchange can be reduced down to the PoSSoWN problem. The idea of the design for the key-exchange is that to recover the shared secret-key, one needs to solve two systems of noisy quadratic system of equations, and recover the contribution of the secret into the shared secret-key from both Alice and Bob. Recovering these contributions is the problem of PoSSoWN. The key-exchange mechanism has been built in such a way that, it uses input from both the users to get a shared key, rather than the trivial way a Key encapsulation works. The hardness of our key exchange can be reduced down to the PoSSoWN problem. The idea of the design for the key-exchange is that to recover the shared secret-key, one needs to solve two systems of noisy quadratic system of equations, and recover the contribution of the secret into the shared secret-key from both Alice and Bob. Recovering these contributions is the problem of PoSSoWN, which is NP-hard. One of the major advantages that CFPKM has, is the cheap communication costs and key sizes. In comparison to similar KEM's based on Learning with Errors, this protocol is able to achieve similar levels of security with much lower values of comparable parameters.

6.6 Why the Scheme Failed

After the first round of submissions, the scheme was broken by Martin Albrecht and his team at Royal Holloway [1]. The main issue that lied in the scheme was not from an algebraic security point of view, but rather a very rudimentary practical fault in the design of the scheme.

Let us have a close look at the design of the shared secret. On Bob's side, the Rounding function computes the shared secret from the B most significant bits of each component of the vector **b3**. The **b3** vector is constructed as follows:

$$\begin{aligned} \mathbf{b_3} &= f(\mathbf{sb}) \odot \mathbf{b1} + \mathbf{e3} \\ &= f(\mathbf{sb}) \odot f(\mathbf{sa}) + f(\mathbf{sb}) \odot \mathbf{e1} + \mathbf{e3} \end{aligned} \tag{6.8}$$

By the choice of the parameters, the errors and the coefficients of the polynomials $f \in \mathbb{F}_q^m[x_1, \ldots, x_n]$ are "small". Now we will show how small are they with respect to evaluations we are dealing with. In particular, we shall demonstrate this for the first set of parameters, i.e., for CFPKM128. The errors are chosen uniformly from a range upper bounded by Range $= 7$. The coefficients of the polynomials are generated by the seed value but are taken modulo Cofsize $= 4096 = 2^{12}$. Now, we see that evaluation of the polynomial $f(\mathbf{sb})$ is upper bounded by

$$(80)^2 \cdot (2^{12} \cdot 7^2) + 80 \cdot (2^{12} \cdot 7) + 2^{12} \leq 2^{31}$$

[1] There is no citation as the issue was pointed out through NIST forums

Thus, the product $f(\mathbf{sb}) \odot \mathbf{e1}$ is upper bounded by 2^{34}. Since the shared secret are the $B = 6$ most significant bits of $\mathbf{b3}$, which is an m dimensional vector where each component is $\log q = 50$ bits, the product $f(\mathbf{sb}) \odot \mathbf{e1}$ has no contribution to the shared secret.

Now let us consider the two public values $\mathbf{b1}$ and $\mathbf{b2}$. Taking the special component wise product of these public vectors we have

$$\begin{aligned} \mathbf{b1} \odot \mathbf{b2} &= (f(\mathbf{sa}) + \mathbf{e1}) \odot (f(\mathbf{sb}) + \mathbf{e2}) \\ &= f(\mathbf{sa}) \odot f(\mathbf{sb}) + f(\mathbf{sa}) \odot \mathbf{e2} + f(\mathbf{sb}) \odot \mathbf{e1} + \mathbf{e1} \odot \mathbf{e2}. \end{aligned} \tag{6.9}$$

Consider the B most significant bits of this product $\mathbf{b1} \odot \mathbf{b2}$. With the same intuition, the B most significant bits have no influence from $f(\mathbf{sb}) \odot \mathbf{e1} + f(\mathbf{sa}) \odot \mathbf{e2}$. Thus, the shared secret is always computed from the product $f(\mathbf{sa}) \odot f(\mathbf{sb})$.

Hence any adversary who can get access to the public values $\mathbf{b1}$ and $\mathbf{b2}$, they can just compute the product and the B most significant bits of it to recover the shared secret.

6.7 Can This Issue be Resolved?

As soon as the scheme was broken, efforts were made to correct the scheme. The first approach taken was to have a fixed support for the polynomials in the public keys. Let us define a square coefficient matrix, say $\mathtt{A}$ of size $m \times m$ over $\mathbb{F}_q$ where the columns of the matrix represent the monomials in the fixed support. By this we are fixing the public polynomials to have the same m monomials. Now, we define the construction of the vectors $\mathbf{b1}, \mathbf{b2}$ and $\mathbf{b3}$.

$$\mathbf{b1} = \mathtt{A} \times \mathbf{sa}' + \mathbf{e1}.$$

$$\mathbf{b2} = \mathbf{sb}' \times \mathtt{A} + \mathbf{e2}.$$

$$\mathbf{b3} = \mathbf{sb}' \times \mathbf{b1} + \mathbf{e3}.$$

where $\mathbf{sa}'$ (and $\mathbf{sb}'$) is the evaluation of the monomials in the fixed support SUP over the secret value $\mathbf{sa}$ (and $\mathbf{sb}$ respectively). On one hand this surely solves the issue of any passive adversary, who has access to the public vectors $\mathbf{b1}$ and $\mathbf{b2}$, being able to recover the hidden secret from $\mathbf{b2} \odot \mathbf{b1}$.

On the other hand, this reflects another instance of the LWE problem and can be viewed as a parameterization of the NewHope key exchange protocol which is based on Ring-LWE [AAB+]. Thus we had to drop this approach.

Another idea was to increase the bound on the coefficient size of the polynomials (i.e q^α) such that we can expect the effect of the product $f(\mathbf{sb}) \odot \mathbf{e1}$ to also influence the B most significant bits. However, according to Corollary 6.2.8 we already have

$$\beta < \left(\frac{k(1-\alpha) - B - 2 - 2\log n}{3} \right),$$

which gives a bound on the range to choose our secret and error required for correct decryption with high probability. Thus increasing the size of the coefficients implies, increasing α. Increasing α, decreases the upper bound on β, which then also decreases the values for the secret and the error. Hence, we don't achieve the expected effect of the product $f(\mathbf{sb}) \odot \mathbf{e1}$ affecting the shared secret.

6.8 Conclusion

We presented a new multivariate key exchange scheme based on the problem of solving a system of noisy multivariate equations, as a submission to the `NIST` Post-quantum standardization process. This key encapsulation was the first of its kind using the `NP`-Hard problem of `PoSSoWN`. We provided a new design that utilizes the use of an error in polynomials in order to blind the information passed over open channels. We provided the scheme along with security analysis against all potential algebraic attacks on it. Finally, we show why because of a structural defect, the scheme was broken.

Bibliography

[AAB+] Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Peter Schwabe Thomas Pöppelmann, and Douglas Stebila. Newhope. Technical report, Technical report, National Institute of Standards and Technology, 2017 ....

[ABD+17] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-kyber algorithm specifications and supporting documentation. *Submission to the NIST post-quantum project*, 9:11, 2017.

[ABD+18] Erdem Alkim, Joppe Bos, Léo Ducas, Patrick Longa, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Chris Peikert, Ananth Raghunathan, Douglas Stebila, et al. Frodokem–learning with errors key encapsulation, 2017. *URL: https://frodokem. org/files/FrodoKEM-specification-20171130. pdf*, 2018.

[AC11] Martin Albrecht and Carlos Cid. Cold boot key recovery by solving polynomial systems with noise. In *International Conference on Applied Cryptography and Network Security*, pages 57–72. Springer, 2011.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *Annual International Cryptology Conference*, pages 595–618. Springer, 2009.

[AFFP11] Martin R Albrecht, Pooya Farshim, Jean-Charles Faugere, and Ludovic Perret. Polly cracker, revisited. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 179–196. Springer, 2011.

[AG11] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. *Automata, languages and programming*, pages 403–415, 2011.

[AGO+17] Koichiro Akiyama, Yasuhiro Goto, Shinya Okumura, Tsuyoshi Takagi, Koji Nuida, Goichiro Hanaoka, Hideo Shimizu, and Yasuhiko Ikematsu. A public-key encryption scheme based on non-linear indeterminate equations (giophantus). *IACR Cryptology ePrint Archive*, 2017:1241, 2017.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 99–108, 1996.

[ALFP] Martin Albrecht, D Lin, Jean-Charles Faugere, and Ludovic Perret. Polynomials with error.

[Bar04] Magali Bardet. *Étude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie.* PhD thesis, 2004.

[BCC+10] Charles Bouillaguet, Hsieh-Chung Chen, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, Adi Shamir, and Bo-Yin Yang. Fast exhaustive search for polynomial systems in f2. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 203–218. Springer, 2010.

[BCD+16] Joppe Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from lwe. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1006–1018. ACM, 2016.

[BCNS15] Joppe W Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Post-quantum key exchange for the tls protocol from the ring learning with errors problem. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 553–570. IEEE, 2015.

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust. The magma algebra system i: The user language. *Journal of Symbolic Computation*, 24(3-4):235–265, 1997.

[BDPA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and GV Assche. The keccak reference. *Submission to NIST (Round 3)*, 13:14–15, 2011.

[Bea02] Stephane Beauregard. Circuit for shor's algorithm using 2n+ 3 qubits. *arXiv preprint quant-ph/0205095*, 2002.

[Ben80] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of statistical physics*, 22(5):563–591, 1980.

[Ber67] Elwyn R Berlekamp. Factoring polynomials over finite fields. *Bell System Technical Journal*, 46(8):1853–1859, 1967.

[Bet11] Luk Bettale. *Cryptanalyse algébrique: outils et applications*. PhD thesis, Paris 6, 2011.

[BFMR11] Charles Bouillaguet, Pierre-Alain Fouque, and Gilles Macario-Rat. Practical key-recovery for all possible parameters of sflash. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 667–685. Springer, 2011.

[BFP08] Luk Bettale, Jean-Charles Faugère, and Ludovic Perret. Cryptanalysis of the trms signature scheme of pkc'05. In *International Conference on Cryptology in Africa*, pages 143–155. Springer, 2008.

[BFP09] Luk Bettale, Jean-Charles Faugere, and Ludovic Perret. Hybrid approach for solving multivariate systems over finite fields. *Journal of Mathematical Cryptology*, 3(3):177–197, 2009.

[BFP12] Luk Bettale, Jean-Charles Faugère, and Ludovic Perret. Solving polynomial systems over finite fields: improved analysis of the hybrid approach. In *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*, pages 67–74, 2012.

[BFP13] Luk Bettale, Jean-Charles Faugère, and Ludovic Perret. Cryptanalysis of hfe, multi-hfe and variants for odd and even characteristic. *Designs, Codes and Cryptography*, 69(1):1–52, 2013.

[BFS04] Magali Bardet, Jean-Charles Faugere, and Bruno Salvy. On the complexity of gröbner basis computation of semi-regular overdetermined algebraic equations. In *Proceedings of the International Conference on Polynomial System Solving*, pages 71–74, 2004.

[BFS15] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. On the complexity of the f5 gröbner basis algorithm. *Journal of Symbolic Computation*, 70:49–70, 2015.

[BFSS13] Magali Bardet, Jean-Charles Faugère, Bruno Salvy, and Pierre-Jean Spaenlehauer. On the complexity of solving quadratic boolean systems. *Journal of Complexity*, 29(1):53–75, 2013.

[BFSY05] Magali Bardet, Jean-Charles Faugere, Bruno Salvy, and Bo-Yin Yang. Asymptotic behaviour of the index of regularity of quadratic semi-regular polynomial systems. In *The Effective Methods in Algebraic Geometry Conference (MEGA'05)(P. Gianni, ed.)*, pages 1–14. Citeseer, 2005.

[BGJT14] Razvan Barbulescu, Pierrick Gaudry, Antoine Joux, and Emmanuel Thomé. A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–16. Springer, 2014.

[BGML+18] Sauvik Bhattacharya, Oscar Garcia-Morchon, Thijs Laarhoven, Ronald Rietman, Markku-Juhani O Saarinen, Ludo Tolhuizen, and Zhenfei Zhang. Round5- compact and fast post-quantum public-key encryption. *IACR Cryptology ePrint Archive*, 2018:725, 2018.

[BL17] Daniel J Bernstein and Tanja Lange. Post-quantum cryptography. *Nature*, 549(7671):188–194, 2017.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 575–584, 2013.

[BPSV19] Ward Beullens, Bart Preneel, Alan Szepieniec, and Frederik Vercauteren. Luov. *Csrc. nist. gov*, 2019.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 409–426. Springer, 2006.

[BR11] Elaine Barker and Allen Roginsky. Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths. *NIST Special Publication*, 800:131A, 2011.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. *Journal of CRYPTOLOGY*, 4(1):3–72, 1991.

[Buc65] Bruno Buchberger. An algorithm for finding the base elements of the residual class ring after a zero-dimensional polynomial ideal. *PhD thesis, Universitat Insbruck*, 1965.

[Buc76] Bruno Buchberger. A theoretical basis for the reduction of polynomials to canonical forms. *SIGSAM Bull.*, 10:19–29, 08 1976.

[BW98] Th Becker and V Weispfennig. H. kredel (1998): Gröbner bases: a computational approach to commutative algebra, corr. 2. print, 1998.

[BWP05] An Braeken, Christopher Wolf, and Bart Preneel. A study of the security of unbalanced oil and vinegar signature schemes. In *Cryptographers' Track at the RSA Conference*, pages 29–43. Springer, 2005.

[Car96] PA Carl. Tale of two sieves. *Notices American Mathematical Society*, 43(12):1473–85, 1996.

[CB07] Nicolas T Courtois and Gregory V Bard. Algebraic cryptanalysis of the data encryption standard. In *IMA International Conference on Cryptography and Coding*, pages 152–169. Springer, 2007.

[CCJ+16] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. *Report on post-quantum cryptography*. US Department of Commerce, National Institute of Standards and Technology, 2016.

[CDF03] Nicolas T Courtois, Magnus Daum, and Patrick Felke. On the security of hfe, hfev-and quartz. In *International Workshop on Public Key Cryptography*, pages 337–350. Springer, 2003.

[CFMR+17] Antoine Casanova, Jean-Charles Faugère, Gilles Macario-Rat, Jacques Patarin, Ludovic Perret, and Jocelyn Ryckeghem. Gemss: A great multivariate short signature. *Submission to NIST*, 2017.

[CFP17] Olive Chakraborty, Jean-Charles Faugère, and Ludovic Perret. Cfpkm: A key encapsulation mechanism based on solving system of non-linear multivariate polynomials. 2017.

[CKPS00] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 392–407. Springer, 2000.

[CLO06] David A Cox, John Little, and Donal O'shea. *Using algebraic geometry*, volume 185. Springer Science & Business Media, 2006.

[CLO15] David Cox, John Little, and Donal O'Shea. Ideals, varieties, and algorithms. 2015.

[CLRS09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[CM97] Robert S Coulter and Rex W Matthews. Planar functions and planes of lenz-barlotti class ii. *Designs, Codes and Cryptography*, 10(2):167–184, 1997.

[Coc73] Clifford C Cocks. A note on non-secret encryption. *CESG Memo*, 1973.

[Cou01] Nicolas T Courtois. The security of hidden field equations (hfe). In *Cryptographers' Track at the RSA Conference*, pages 266–281. Springer, 2001.

[CST17] Ryann Cartor and Daniel Smith-Tone. An updated security analysis of pflash. In *International Workshop on Post-Quantum Cryptography*, pages 241–254. Springer, 2017.

[CST18] Ryann Cartor and Daniel Smith-Tone. Eflash: A new multivariate encryption scheme. In *International Conference on Selected Areas in Cryptography*, pages 281–299. Springer, 2018.

[CSTV] Daniel Cabarcas, Daniel Smith-Tone, and Javier A Verbel. Practical key recovery attack for zhfe.

[CYST15] Ming-Shing Chen, Bo-Yin Yang, and Daniel Smith-Tone. Pflash-secure asymmetric signatures on smart cards. In *Lightweight Cryptography Workshop*, 2015.

[CZ81] David G Cantor and Hans Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, pages 587–592, 1981.

[Dan15] Quynh H Dang. Secure hash standard. Technical report, 2015.

[DFSS07] Vivien Dubois, Pierre-Alain Fouque, Adi Shamir, and Jacques Stern. Practical cryptanalysis of sflash. In *Annual International Cryptology Conference*, pages 1–12. Springer, 2007.

[DG06] Jintai Ding and Jason E Gower. Inoculating multivariate schemes against differential attacks. In *International Workshop on Public Key Cryptography*, pages 290–301. Springer, 2006.

[DH76a] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[DH76b] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[Din04] Jintai Ding. A new variant of the matsumoto-imai cryptosystem through perturbation. In *International Workshop on Public Key Cryptography*, pages 305–318. Springer, 2004.

[Dir39] Paul Adrien Maurice Dirac. A new notation for quantum mechanics. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 35, pages 416–418. Cambridge University Press, 1939.

[DK12] Jintai Ding and Thorsten Kleinjung. Degree of regularity for hfe minus (hfe-). *JMI: journal of math-for-industry*, 4:97–104, 2012.

[DKRV18] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure kem. In *International Conference on Cryptology in Africa*, pages 282–305. Springer, 2018.

[DO68] Peter Dembowski and Thedore G Ostrom. Planes of order n with collineation groups of ordern 2. *Mathematische Zeitschrift*, 103(3):239–258, 1968.

[DPW14] Jintai Ding, Albrecht Petzoldt, and Lih-Chung Wang. The cubic simple matrix encryption scheme. volume 8772, pages 76–87, 10 2014.

[DR99] Joan Daemen and Vincent Rijmen. The rijndael block cipher: Aes proposal. In *First candidate conference (AeS1)*, pages 343–348, 1999.

[DS05] Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. In *International Conference on Applied Cryptography and Network Security*, pages 164–175. Springer, 2005.

[DXL12] Jintai Ding, Xiang Xie, and Xiaodong Lin. A simple provably secure key exchange scheme based on the learning with errors problem. *IACR Cryptology EPrint Archive*, 2012:688, 2012.

[DYC+07] Jintai Ding, Bo-Yin Yang, Chen-Mou Cheng, Chia-Hsin Owen Chen, and Vivien Dubois. Breaking the symmetry: a way to resist the new differential attack. *IACR Cryptology ePrint Archive*, 2007:366, 2007.

[DZD+19] Jintai Ding, Zheng Zhang, Joshua Deaton, Kurt Schmidt, and F Vishakha. New attacks on lifted unbalanced oil vinegar. In *the 2nd NIST PQC Standardization Conference*, 2019.

[EF14] Christian Eder and Jean-Charles Faugère. A survey on signature-based gr\" obner basis computations. *arXiv preprint arXiv:1404.1774*, 2014.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[Ell70] James H Ellis. The possibility of secure non-secret digital encryption. *UK Communications Electronics Security Group*, page 6, 1970.

[Fau99] Jean-Charles Faugere. A new efficient algorithm for computing gröbner bases (f4). *Journal of pure and applied algebra*, 139(1-3):61–88, 1999.

[Fau02] Jean Charles Faugère. A new efficient algorithm for computing gröbner bases without reduction to zero (f 5). In *Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, pages 75–83. ACM, 2002.

[Fey82] Richard P Feynman. Simulating physics with computers. *Int. J. Theor. Phys*, 21(6/7), 1982.

[FGHR13] Jean-Charles Faugère, Pierrick Gaudry, Louise Huot, and Guénaël Renault. Polynomial systems solving by fast linear algebra. *arXiv preprint arXiv:1304.6039*, 2013.

[FGLM94] JC Faugere, P Gianni, D Lazard, and T Mora. Efficient computation of zero dimensional gröbner bases by change of ordering. *J. Symbolic Computation*, 11:1–000, 1994.

[FGS05] Pierre-Alain Fouque, Louis Granboulan, and Jacques Stern. Differential cryptanalysis for multivariate schemes. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 341–353. Springer, 2005.

[FHK+17] Jean-Charles Faugère, Kelsey Horan, Delaram Kahrobaei, Marc Kaplan, Elham Kashefi, and Ludovic Perret. Fast quantum algorithm for solving multivariate quadratic equations. *arXiv preprint arXiv:1712.07211*, 2017.

[FJ03] Jean-Charles Faugere and Antoine Joux. Algebraic cryptanalysis of hidden field equation (hfe) cryptosystems using gröbner bases. In *Annual International Cryptology Conference*, pages 44–60. Springer, 2003.

[FJPT10] Jean-Charles Faugere, Antoine Joux, Ludovic Perret, and Joana Treger. Cryptanalysis of the hidden matrix cryptosystem. In *International Conference on Cryptology and Information Security in Latin America*, pages 241–254. Springer, 2010.

[FPS09] Jean-Charles Faugère, Ludovic Perret, and Pierre-Jean Spaenlehauer. Algebraic-differential cryptanalysis of des. In *Western European Workshop on Research in Cryptology-WEWoRC*, volume 2009, pages 1–5. Citeseer, 2009.

[FY79] Aviezri S Fraenkel and Yaacov Yesha. Complexity of problems in games, graphs and algebraic equations. *Discrete Applied Mathematics*, 1(1-2):15–30, 1979.

[GG16] Steven D Galbraith and Pierrick Gaudry. Recent progress on the elliptic curve discrete logarithm problem. *Designs, Codes and Cryptography*, 78(1):51–72, 2016.

[GJ79] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[GM88] Rüdiger Gebauer and H Michael Möller. On an installation of buchberger's algorithm. *Journal of Symbolic computation*, 6(2-3):275–286, 1988.

[GM02] Henri Gilbert and Marine Minier. Cryptanalysis of sflash. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 288–298. Springer, 2002.

[GMSS99] Oded Goldreich, Daniele Micciancio, Shmuel Safra, and J-P Seifert. Approximating shortest lattice vectors is not harder than approximating closest lattice vectors. *Information Processing Letters*, 71(2):55–61, 1999.

[Gro96] Lov K Grover. A fast quantum mechanical algorithm for database search. *arXiv preprint quant-ph/9605043*, 1996.

[Gu16] Chunsheng Gu. Cryptanalysis of simple matrix scheme for encryption. *IACR Cryptology ePrint Archive*, 2016:1075, 2016.

[HRSS16] Andreas Hülsing, Joost Rijneveld, Simona Samardjiska, and Peter Schwabe. From 5-pass mq-based identification to mq-based signatures. *IACR Cryptology ePrint Archive*, 2016:708, 2016.

[IPST+18] Yasuhiko Ikematsu, Ray Perlner, Daniel Smith-Tone, Tsuyoshi Takagi, and Jeremy Vates. Hferp-a new multivariate encryption scheme. In *International Conference on Post-Quantum Cryptography*, pages 396–416. Springer, 2018.

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[KPG99] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced oil and vinegar signature schemes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 206–222. Springer, 1999.

[KS98] Aviad Kipnis and Adi Shamir. Cryptanalysis of the oil and vinegar signature scheme. In *Annual International Cryptology Conference*, pages 257–266. Springer, 1998.

[KSWH98] John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Side channel cryptanalysis of product ciphers. In *European Symposium on Research in Computer Security*, pages 97–110. Springer, 1998.

[Laz83] Daniel Lazard. Gröbner bases, gaussian elimination and resolution of systems of algebraic equations. In *European Conference on Computer Algebra*, pages 146–156. Springer, 1983.

[LM09] Vadim Lyubashevsky and Daniele Micciancio. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In *Annual International Cryptology Conference*, pages 577–594. Springer, 2009.

[LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for lwe-based encryption. In *Cryptographers' Track at the RSA Conference*, pages 319–339. Springer, 2011.

[LPT+17] Daniel Lokshtanov, Ramamohan Paturi, Suguru Tamaki, Ryan Williams, and Huacheng Yu. Beating brute force for systems of polynomial equations over finite fields. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2190–2202. SIAM, 2017.

[Mac02] Francis Sowerby Macaulay. Some formulae in elimination. *Proceedings of the London Mathematical Society*, 1(1):3–27, 1902.

[Mat93] Mitsuru Matsui. Linear cryptanalysis method for des cipher. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 386–397. Springer, 1993.

[McC90] Kevin S McCurley. The discrete logarithm problem. In *Proc. of Symp. in Applied Math*, volume 42, pages 49–74. USA, 1990.

[Mer78] Ralph C Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21(4):294–299, 1978.

[MI88] Tsutomu Matsumoto and Hideki Imai. Public quadratic polynomial-tuples for efficient signature-verification and message-encryption. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 419–453. Springer, 1988.

[MM82] Ernst W Mayr and Albert R Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in mathematics*, 46(3):305–329, 1982.

[Mon94] Peter L Montgomery. A survey of modern integer factorization algorithms. *CWI quarterly*, 7(4):337–366, 1994.

[MPST16] Dustin Moody, Ray Perlner, and Daniel Smith-Tone. Key recovery attack on the cubic abc simple matrix multivariate encryption scheme. In *International Conference on Selected Areas in Cryptography*, pages 543–558. Springer, 2016.

[MV04] David A McGrew and John Viega. The security and performance of the galois/counter mode (gcm) of operation. In *International Conference on Cryptology in India*, pages 343–355. Springer, 2004.

[NIS92] CORPORATE NIST. The digital signature standard. *Communications of the ACM*, 35(7):36–40, 1992.

[ØFRC20] Morten Øygarden, Patrick Felke, Håvard Raddum, and Carlos Cid. Cryptanalysis of the multivariate encryption scheme eflash. In *Cryptographers' Track at the RSA Conference*, pages 85–105. Springer, 2020.

[Pat95] Jacques Patarin. Cryptanalysis of the matsumoto and imai public key scheme of eurocrypt'88. In *Annual International Cryptology Conference*, pages 248–261. Springer, 1995.

[Pat96] Jacques Patarin. Hidden fields equations (hfe) and isomorphisms of polynomials (ip): Two new families of asymmetric algorithms. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 33–48. Springer, 1996.

[Pat97] Jacques Patarin. The oil and vinegar signature scheme. In *Dagstuhl Workshop on Cryptography September, 1997*, 1997.

[PBD14] Jaiberth Porras, John Baena, and Jintai Ding. Zhfe, a new multivariate public key encryption scheme. In *International workshop on post-quantum cryptography*, pages 229–245. Springer, 2014.

[PCG01a] Jacques Patarin, Nicolas Courtois, and Louis Goubin. Flash, a fast multivariate signature algorithm. In *Cryptographers' Track at the RSA Conference*, pages 298–307. Springer, 2001.

[PCG01b] Jacques Patarin, Nicolas Courtois, and Louis Goubin. Quartz, 128-bit long digital signatures. In *Cryptographers' Track at the RSA Conference*, pages 282–297. Springer, 2001.

[PCY+15] A Petzoldt, MS Chen, BY Yang, C Tao, and J Ding. Design principles for hfevbased signature schemes. asiacrypt 2015-part 1, lncs vol. 9452, 2015.

[PCY+17] A Petzoldt, MS Chen, BY Yang, C Tao, and J Ding. Gui documentation,https://csrc. nist.gov/projects/post-quantum-cryptography/round-1-submissions, 2017.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 333–342, 2009.

[Pei14] Chris Peikert. Lattice cryptography for the internet. In *International Workshop on Post-Quantum Cryptography*, pages 197–219. Springer, 2014.

[PGC98] Jacques Patarin, Louis Goubin, and Nicolas Courtois. C-+* and hm: Variations around two schemes of t. matsumoto and h. imai. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 35–50. Springer, 1998.

[Pol78] John M Pollard. Monte carlo methods for index computation. *Mathematics of computation*, 32(143):918–924, 1978.

[Pol00] John M Pollard. Kangaroos, monopoly and discrete logarithms. *Journal of cryptology*, 13(4):437–447, 2000.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

[RSA78] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[Sch95] Benjamin Schumacher. Quantum coding. *Physical Review A*, 51(4):2738, 1995.

[SDP16] Alan Szepieniec, Jintai Ding, and Bart Preneel. Extension field cancellation: A new central trapdoor for multivariate quadratic systems. In *International workshop on post-quantum cryptography*, pages 182–196. Springer, 2016.

[Sha49] Claude E Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28(4):656–715, 1949.

[Sha93] Adi Shamir. Efficient signature schemes based on birational permutations. In *Annual International Cryptology Conference*, pages 1–12. Springer, 1993.

[Sho99] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[SK99] Adi Shamir and Aviad Kipnis. Cryptanalysis of the hfe public key cryptosystem. In *Advances in Cryptology, Proceedings of Crypto*, volume 99, 1999.

[SS98] Joseph H Silverman and Joe Suzuki. Elliptic curve discrete logarithms and the index calculus. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 110–125. Springer, 1998.

[Sto00] Arne Storjohann. *Algorithms for matrix canonical forms*. PhD thesis, ETH Zurich, 2000.

[SW16] Peter Schwabe and Bas Westerbaan. Solving binary mq with grover's algorithm. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 303–322. Springer, 2016.

[TDTD13] Chengdong Tao, Adama Diene, Shaohua Tang, and Jintai Ding. Simple matrix scheme for encryption. In *International Workshop on Post-Quantum Cryptography*, pages 231–242. Springer, 2013.

[VOW99] Paul C Van Oorschot and Michael J Wiener. Parallel collision search with cryptanalytic applications. *Journal of cryptology*, 12(1):1–28, 1999.

[VST17] Jeremy Vates and Daniel Smith-Tone. Key recovery attack for all parameters of hfe. In *International Workshop on Post-Quantum Cryptography*, pages 272–288. Springer, 2017.

[Wil12] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 887–898, 2012.

Appendices

Appendix A

EFC-Source Code

This is the MAGMA code for EFC.

```
// Uncomment to run the challenge parameter
/*
//Challenge 1
n := 83; q :=2; Frobenius:=0, a := 10;
aa1 := Random(1,a);
*/
/*
//Challenge 2
n := 83; q :=2; Frobenius:=1, a := 8;
aa1 := Random(1,a);
*/
/*
//Challenge 3
n := 56; q :=3; Frobenius:=0, a := 10;
aa1 := Random(1,10);
*/

// Some more global variables
degmul := 2; // the degree to be set for the degree of the
   multipliers
degreq := 2;//the degree after which we consider the equations for
    constructing our matrix of coefficients and computing the
   kernel,
Frodeg := n-1;//the degree 2^frodeg upto which which we consider
   the frobenius powers of the polynomials to consider all
   solutions




frac := 0.35;
Hybrid :=0;
Tval :=Ceiling(n*frac);
\oc{move this part to the special part of hybrid approach}
```

```
SetMemoryLimit(0);

t1 := Cputime();



SetVerbose("Groebner",1);


if Hybrid eq 1 then
  print "Hybrid mode on\n","frac=",frac;
end if;
print "n=",n,"a=",a,"aa1=",aa1,"q=",q, "Fro=",Frobenius,"Proj=",
   Proj,"degreq=",degreq ;

print "degmul=",degmul,"Frodeg=",Frodeg;


//defining the base field
k := 1;
BaseF := GF(q^k);
//defining the extension field
Rp := IrreduciblePolynomial(BaseF,n);
ExtF<w> := ext<BaseF|Rp>;
Bas := Basis(ExtF);
SetPowerPrinting(ExtF, false);

// Polynomial ring over the base field with n indeterminates
P1<[x]> := PolynomialRing(BaseF,n, "grevlex");

// Polynomial ring over the extension field
P2 := ChangeRing(P1,ExtF);

//defining the field eqs
fieldeq1 := [P1!x[i]^q-P1!x[i]: i in [1..n]];
fieldeq2 := [P2!x[i]^q-P2!x[i]: i in [1..n]];


//The Projection modifier, 0 to set it to false, 1 to set true
Proj :=0;
//the secret matrices
if Proj eq 1 then
  A := RandomMatrix(BaseF, n,n);
  B := RandomMatrix(BaseF, n,n);
  while (A eq B) or (Rank(A) ne (n-1)) or (Rank(B) ne (n-1)) or (
   Kernel(A) eq Kernel(B))do // just to ensure that we dont get
   the same set of equations again
    A := RandomMatrix(BaseF, n,n);
    B := RandomMatrix(BaseF, n,n);
  end while;
else
```

```
  A := Random(GL(n,BaseF));
  B := Random(GL(n,BaseF));
end if;



// This function reduces a monomial by the field eqs
RedChar := function(P,mon)
  newmon := P!1;
  ex := Exponents(mon);
  if q eq 2 then
    for i in [1..n] do
    if Gcd(P!mon,P!x[i]) ne 1 then
      newmon := newmon * P!x[i];
    end if;
    end for;
  else
    for i in [1..n] do
      if (ex[i] mod (q-1)) eq 1 then
        newmon := newmon * P!x[i];
      elif (ex[i] mod q) eq 0 then
        while ((ex[i] mod q) eq 0) and (ex[i] ge q) do
          ex[i] := Integers()!(ex[i]/q);
        end while;
        newmon := newmon * P!x[i]^(Integers()!(ex[i]));
      else newmon := newmon * P!x[i]^(ex[i] mod q);
      end if;
    end for;
  end if;
  return newmon;
end function;


LinearTransform := function(LT,vec) //here the vec should be a
   column vector
  Lt := ChangeRing(LT,P1);
  retvec := Lt*vec;
  return retvec;
end function;

LinearTransformInv := function(LT,vec)
  vec_mat := Matrix(P1,NumberOfRows(LT),1,vec);
  inv := (LT^(-1));
  inv := ChangeRing(inv,P1);
  retvec := inv*vec_mat;
  return retvec;
end function;

//convert a seq to a column of a matrix
LC :=  function(d,mat1,j)
  tempseq := Eltseq(d);
```

```
  for k in [1..n] do
    mat1[j][k] := tempseq[k];
  end for;
  return mat1;
end function;

// function to convert a system of n multivariate base field polys to one poly in ext field of the form \sum(f_i*w^i)
Ext_pol := function(pol_list)
  temp := P2!0;
  for i in [1..n] do
    temp := temp + P2!pol_list[i][1]*Bas[i];
  end for;
  return temp;
end function;




//function to convert one poly in ext field to system of n multivariate polys in base field..returned as a single column matrix
Pol_mat := function(f)
  rows := #Monomials(f);
  colms := #Bas;
  mat := Matrix(BaseF,rows,colms,[0 : i in [1..rows*colms]]);
  for i in [1..rows] do
    c := MonomialCoefficient(f,Monomials(f)[i]);
    mat := LC(c,mat,i);
  end for;
  Mons :=[];
  for i in [1..rows] do
    Mons[i] :=  RedChar(P1,Monomials(f)[i]);
  end for;
  final_pols :=[];
  for j in [1..n] do
    temp := P1!0;
    for i in [1..rows] do
       temp := temp + P1!(Mons[i])*mat[i][j];
    end for;
    final_pols := Append(final_pols,temp);
  end for;
  return Matrix(P1,n,1,final_pols);
end function;

S := IdentityMatrix(BaseF,n);
//T := Random(GL(2*n,BaseF));


////////////////////////////////////////only special case ////////////////////////
```

```
T := IdentityMatrix(BaseF,2*n);


//taking input vector variables x[1],....,x[n]
main_vec_enc := Matrix(P1,n,1,[x[i] : i in [1..n]]) ;

//applying projection modifier
if Proj eq 1 then
  main_vec_enc := Matrix(P1,n,1,[x[i] : i in [1..n-1]] cat [0]);
end if;

//doing linear transformation S over input vector
print "time for S map";
time inpvec_enc := LinearTransform(S,main_vec_enc);

//Generating message to be encrypted

if Proj eq 1 then
  s := [Random(BaseF) : i in [1..n-1]] cat [0];
  while s eq [0: i in [1..n]] do
    s := [Random(BaseF) : i in [1..n-1]] cat [0];
  end while;
else
  s := [Random(BaseF) : i in [1..n]];
  while s eq [0: i in [1..n]] do
    s := [Random(BaseF) : i in [1..n]];
  end while;
end if;
s_mat := Matrix(BaseF, n,1,s);
print "Msg to be encrypted =",s;


// function to compute power of a ext field multivariate rep of
   the varaible \chi
Ext_var_pow := function(chi,i)
        kai := P2!1;
        I  := Intseq(i,q);
    if q eq 2 then
      while Index(I,1) ne 0 do
    //for j in [1..#I] do
        i1 := Index(I,1)-1;
        //print j, i1;
        phi := P2!0;
              for k in [1..n] do
                  phi := phi + (P2!(P1.k))*(Bas[k]^(q^i1));
              end for;
              kai := kai * phi;
              I[i1+1] :=0;
            end while;
        elif q eq 3 then
      for j in [1..q-1] do
```

```
        while Index(I,j) ne 0 do
        //for j in [1..#I] do
          i1 := Index(I,j)-1;
        //print j, i1;
          if j eq 1 then
            phi := P2!0;
                  for k in [1..n] do //for the x[1]^j part
                      phi := phi + (P2!(P1.k)^j)*(Bas[k]^(j*q^i1))
    ;
                  end for;
                elif j eq 2 then
                  phi := P2!0;
                  for k in [1..n] do //for the x[1]^j part
                      phi := phi + (P2!(P1.k)^j)*(Bas[k]^(j*q^i1))
    ;
                  end for;
                  for k in [1..n] do //for the multiplicative part
                    for l in [k+1..n] do
                    phi := phi + (P2!(P1.k)*P2!(P1.l)*(j*Bas[k]*
    Bas[l])^(q^i1)) ;
                    end for;
                  end for;
                end if;
              kai := kai * phi;
              I[i1+1] :=0;
              end while;
      end for;
     end if;
  return NormalForm(kai,fieldeq2);
end function;

//////////////////////////////////////////////////////////////


x_vec := inpvec_enc;
chi_x1 := Ext_pol(x_vec);

AA := [Random(ExtF) : i in [1..n]];
BB := [Random(ExtF) : i in [1..n]];

F_1Ex := P2!0;
F_2Ex := P2!0;
if Frobenius eq 0 then
  print "no frobenius";
  for i in [0..n-1] do
    F_1Ex := F_1Ex + AA[i+1]*Ext_var_pow(chi_x1,(q^i + 1));
    F_2Ex := F_2Ex + BB[i+1]*Ext_var_pow(chi_x1,(q^i + 1));
  end for;
else
  print "frobenius applied";
  for i in [0..n-1] do
```

```
    F_1Ex := F_1Ex + AA[i+1]*Ext_var_pow(chi_x1,(q^i + 1)) + (AA[i+1])^3*Ext_var_pow(chi_x1,(q^(i+1) + q^i));
    F_2Ex := F_2Ex + BB[i+1]*Ext_var_pow(chi_x1,(q^i + 1))+ (BB[i+1])^3*Ext_var_pow(chi_x1,(q^(i+1) + q^i));
  end for;
end if;


AA1 := [ExtF!1 : i in [1..aa1]];
AA1 := [ExtF!Random(ExtF) : i in [1..aa1]];
//t1 := Cputime();
pi_F1Ex := F_1Ex;
for i in [1..aa1] do
  pi_F1Ex := pi_F1Ex + AA1[i]*(F_1Ex)^(q^i);
end for;
pi_F1Ex := NormalForm(pi_F1Ex,fieldeq2);
//t2 := Cputime(t1); print "time for F1=",t2;


pi_F2Ex := F_2Ex;
BB1 := [ExtF!1 :  i in [1..aa2]];
BB1 := [ExtF!Random(ExtF) :  i in [1..aa2]];
//t1 := Cputime();
for i in [1..aa2] do
  pi_F2Ex := pi_F2Ex + BB1[i]*(F_2Ex)^(q^i);
end for;
pi_F2Ex := NormalForm(pi_F2Ex,fieldeq2);
//t2 := Cputime(t1); print "time for F1=",t2;



time pi_F1bas := Pol_mat(pi_F1Ex);
cipher1 := [Evaluate(pi_F1bas[i][1],s) : i in [1..n]];

time pi_F2bas := Pol_mat(pi_F2Ex);
cipher2 := [Evaluate(pi_F2bas[i][1],s) : i in [1..n]];

C7 := [pi_F1bas[i][1] - cipher1[i] : i in [1..n]];
C8 := [pi_F2bas[i][1] - cipher2[i] : i in [1..n]];
M1 := Ext_pol(Matrix(GF(q),n,1,cipher1));
M2 := Ext_pol(Matrix(GF(q),n,1,cipher2));

C9  := C7 cat C8 cat fieldeq1;
print "\n Doing GB calculation on Puiblic keys\n ";
t1 := Cputime();
G := GroebnerBasis(C9);
t2 := Cputime(t1);
//G;
print "Old GB time is =", t2;


```

```
//// Now will do the process for recovering the intermediate
    equations

Seq := function(M)
  temp :=[];
  for i in [1..NumberOfColumns(M)] do
    temp := Append(temp,M[i]);
  end for;
  return temp;
end function;


NoOfOnes := function(list)
  count :=0;
  if q eq 2 then
    for i in [1..#list] do
      if list[i] eq 1 then
        count := count +1;
      end if;
    end for;
    return count;
  else
    for i in [1..#list] do
      //if list[i] eq 1 then
        count := count +list[i];
      //end if;
    end for;
    return count;
  end if;
end function;

DIV := function(a,b)
  rem := a mod b;
  while ((a div b) ge b) do
    a := a div b;
  end while;
  if ((a div b) + rem) ge q^n then
    return ((((a div b) + rem) mod q^n) + (((a div b) + rem) div q
    ^n));
  else
    return (a div b) + rem;
  end if;
end function;



deg1 := (q^(n-1) + 1)*q^(aa1);// degree of pi_F1Ex
deg2 := (q^(n-1) + 1)*q^(aa2);// degree of pi_F2Ex
maxdeg:= (Max(deg1,deg2))*(q^Frodeg) ;
mindeg := Min(deg1,deg2);
//t1 := Cputime();
```

```
novar :=[];
curr_k:= 1;
prevtmpdeg := 0;
for k in [1..2*(Frodeg+1)]  do
  tmpnovar := [0];
  if IsEven(k) then //  for the multiplicand of F_2 and its
   frobenius powers the multipicand is bounded by the degree of
   deg1
    if q eq 2 then
      tmpdeg1 := q^(n-1)+q^(n-2);
    elif q eq 3 then
      tmpdeg1 :=deg1*(q^Frodeg);
    end if;
    tmpdeg2 := deg2*(q^(k/2-1));
  else  //  for the multiplicand of F_1 and its frobenius powers
   the multipicand is bounded by the degree of deg2
    if q eq 2 then
      tmpdeg1 := q^(n-1)+q^(n-2);
    elif q eq 3 then
      tmpdeg1 :=deg2*(q^Frodeg);
    end if;
    tmpdeg2 := deg1*(q^((k+1)/2-1));
   end if;
  if IsEven(k) then
    for j in [(k/2-1)..Ceiling(Log(q,tmpdeg1))] do
      if DIV(q^j,q^n) notin tmpnovar then
        Append(~tmpnovar,DIV(q^j,q^n));
      end if;
      if degmul gt 1 then
        for i in [0..j] do
          if q^j+q^i le tmpdeg1 then
            if DIV(q^j+q^i,q^n) notin tmpnovar then
              Append(~tmpnovar,DIV(q^j+q^i,q^n));
            end if;
          end if;
        end for;
      end if;
    end for;
  else
    for j in [((k+1)/2-1)..Ceiling(Log(q,tmpdeg1))] do
      if DIV(q^j,q^n) notin tmpnovar then
        Append(~tmpnovar,DIV(q^j,q^n));
      end if;
      if degmul gt 1 then
        for i in [0..j] do
          if q^j+q^i le tmpdeg1 then
            if DIV(q^j+q^i,q^n) notin tmpnovar then
              Append(~tmpnovar,DIV(q^j+q^i,q^n));
            end if;
          end if;
        end for;
```

```
      end if;
    end for;
  end if;
  prevtmpdeg := Max(tmpdeg1,prevtmpdeg);
  if prevtmpdeg eq tmpdeg1 then
    curr_k := k;
  end if;
  //if syzygy eq 0 then
  //  Exclude(~tmpnovar,tmpdeg2);
  //end if;
  Sort(~tmpnovar);
    Append(~novar,tmpnovar);
end for;


novar3 :=[];
degc_x := deg1*(q^Frodeg)+deg2*(q^Frodeg);

////////declairing the number of variabels
numvar := 0;
for i in [1..#novar] do
  numvar := numvar + #novar[i];
end for;
print "Done computing number of variables=",numvar, "time=",t2;

P4 := PolynomialRing(ExtF,numvar,"glex");
P5<Y> := PolynomialRing(P4);



ModofY := function(f)
  tmp_f := Parent(f)!0;
  tmpSup,tmpCof := Support(f);
  for i in [1..#tmpSup] do
    tmp_f := tmp_f + tmpCof[i]*Y^(((tmpSup[i] mod (q^n))));//
   tmpSup[i] div (2^n)) +
  end for;
  return tmp_f;
end function;


//create new variabels for the coefficients
AA_var := [];
for i in [1..#AA] do
  Append(~AA_var,"A" cat IntegerToString(i));
end for;

BB_var := [];
for i in [1..#BB] do
  Append(~BB_var,"B" cat IntegerToString(i));
end for;
```

```
469
470 new_var := AA_var cat BB_var;
471
472
473
474 F_1ex := P5!0;
475 F_2ex := P5!0;
476 count :=0;
477 F_1exS := [];
478 F_1exSup := [];
479
480 for i in [0..n-1] do
481   Append(~F_1exS,q^i + 1);
482   Append(~F_1exSup,P4!AA[i+1]);
483 end for;
484
485 F_2exS := [];
486 F_2exSup := [];
487
488 for i in [0..n-1] do
489   Append(~F_2exS,q^i + 1);
490   Append(~F_2exSup,P4!BB[i+1]);
491 end for;
492
493 t1 := Cputime();
494 pi_F1exS := F_1exS;
495 pi_F1exSup := F_1exSup;
496 pi_F1ex := F_1ex;
497 for i in [1..aa1] do
498   for j in [1..#F_1exS] do
499     if ( DIV(F_1exS[j]*(q^i),(q^n))) notin pi_F1exS then
500       Append(~pi_F1exS,( DIV(F_1exS[j]*(q^i), (q^n))));
501       Append(~pi_F1exSup, AA1[i]*F_1exSup[j]^(q^i));
502       ParallelSort(~pi_F1exS,~pi_F1exSup);
503     else
504       k := Index(pi_F1exS,( DIV(F_1exS[j]*(q^i),(q^n))));
505       pi_F1exSup[k] := pi_F1exSup[k] + AA1[i]*F_1exSup[j]^(q^i);
506     end if;
507   end for;
508 end for;
509 Append(~pi_F1exS,Degree(P5!M1));
510 Append(~pi_F1exSup,P5!-M1);
511 ParallelSort(~pi_F1exS,~pi_F1exSup);
512
513 t2 := Cputime(t1);print "Time pi_f1=",t2;
514 t1 := Cputime();
515 pi_F2ex := F_2ex;
516
517 pi_F2exS := F_2exS;
518 pi_F2exSup := F_2exSup;
519 for i in [1..aa2] do
```

```
  for j in [1..#F_2exS] do
    if (DIV(F_2exS[j]*(q^i), (q^n))) notin pi_F2exS then
      Append(~pi_F2exS,(DIV(F_2exS[j]*(q^i),(q^n))));
      Append(~pi_F2exSup, BB1[i]*F_2exSup[j]^(q^i));
      ParallelSort(~pi_F2exS,~pi_F2exSup);
    else
      k := Index(pi_F2exS,(DIV(F_2exS[j]*(q^i),(q^n))));
      pi_F2exSup[k] := pi_F2exSup[k] + BB1[i]*F_2exSup[j]^(q^i);
    end if;
  end for;
end for;
Append(~pi_F2exS,Degree(P5!M2));
Append(~pi_F2exSup,P5!-M2);
ParallelSort(~pi_F2exS,~pi_F2exSup);

t2 := Cputime(t1);print "Time pi_f2=",t2;

s1 :=  Ext_pol(s_mat);
//////using univariate
t1 := Cputime();
mul :=[];
count :=0;

SSmul :=[];
Supmul := [];

for j in [1..#novar] do //the last +1 is for c_y

  tmpmul :=[];
  tmpSS :=[];
  for i in [1..#novar[j]] do
    tmpSS := Append(tmpSS,novar[j][i]);
    tmpmul := Append(tmpmul,P4.(i+count));
  end for;
  count := count + #novar[j];
  Append(~SSmul,tmpSS);
  Append(~Supmul,tmpmul);
end for;

count1 := count;

t2 := Cputime(t1);print "Time for novar =",t2;


////////////////////////////////////////////////////////////////////////////////


t1 := Cputime();

T1 :=P5!0;T2 := P5!0;
```

```
if novar3 ne [] then
  S3 := SSmul[#novar];
  SC := Supmul[#novar];
else S3,SC := Support(P5!0);
end if;

t1 := Cputime();
SSact :=[];
Supact :=[];


tmpS1 := pi_F1exS;
tmpSup1 := pi_F1exSup;
tmpSup2 := pi_F2exSup;
tmpS2 := pi_F2exS;


for k in [0..Frodeg] do
  tmpS11 := [];
  tmpSup11 := [];
  for i in [1..#tmpS1] do
    if (DIV((tmpS1[i]*(q^k)),(q^n))) notin tmpS11 then
      Append(~tmpS11,(DIV((tmpS1[i]*(q^k)),(q^n)) ));
      Append(~tmpSup11,ExtF!tmpSup1[i]^(q^k));
      ParallelSort(~tmpS11,~tmpSup11);
    else
      l := Index(tmpS11,(DIV((tmpS1[i]*(q^k)),(q^n))));
      tmpSup11[l] := tmpSup11[l] + ExtF!tmpSup1[i]^(q^k);
    end if;
  end for;
  Append(~SSact,tmpS11);
  Append(~Supact,tmpSup11);
  tmpS22 := [];
  tmpSup22 := [];
  for i in [1..#tmpS2] do
    if (DIV((tmpS2[i]*(q^k)),(q^n)) ) notin tmpS22 then
      Append(~tmpS22,(DIV((tmpS2[i]*(q^k)), (q^n)) ));
      Append(~tmpSup22,ExtF!tmpSup2[i]^(q^k));
      ParallelSort(~tmpS22,~tmpSup22);
    else
      l := Index(tmpS22,(DIV((tmpS2[i]*(q^k)), (q^n))));
      tmpSup22[l] := tmpSup22[l] + ExtF!tmpSup2[i]^(q^k);
    end if;
  end for;
  Append(~SSact,tmpS22);
  Append(~Supact,tmpSup22);
end for;
t2 := Cputime(t1);
print "Time taken for making all the frobenius powers of public
   keys, time=", t2;

```

```
620
621
622 TT11 := [];
623 SS11 := [];
624 SS11act := [];
625 t1 := Cputime();
626 for k in [1..2*(Frodeg+1)] do
627   for i in [1..#SSmul[k]] do
628     for j in [1..#SSact[k]] do
629       if (DIV((SSmul[k][i]+SSact[k][j]),(q^n))) notin SS11 then
630         Append(~TT11,Supmul[k][i]*Supact[k][j]);
631         Append(~SS11,( DIV((SSmul[k][i]+SSact[k][j]),(q^n))));
632         if ( DIV((SSmul[k][i]+SSact[k][j]),(q^n))) gt q^n then
633           print "true",(SSmul[k][i]+SSact[k][j]),( DIV((SSmul[k][i
    ]+SSact[k][j]),(q^n)));
634         end if;
635         ParallelSort(~SS11,~TT11);
636       else
637         l := Index(SS11,( DIV((SSmul[k][i]+SSact[k][j]),(q^n))));
638         TT11[l] := TT11[l] + Supmul[k][i]*Supact[k][j];
639       end if;
640     end for;
641   end for;
642 end for;
643 t2 := Cputime(t1);
644 print "Time for multiplyuing the multiplicand and public keys and
     frobenius, time=",t2;
645 #SS11;
646
647 Eqs4 := [];
648 Supsup :=[];
649 for i in [1..#SS11] do
650   if NoOfOnes(Intseq(SS11[i],q)) gt degreq then
651     Append(~Supsup,SS11[i]);Append(~Eqs4,TT11[i]);
652   end if;
653 end for;
654
655 print "Done making the linear eqs";
656 Eqs := Eqs4;
657
658
659 if Index(Eqs,0) gt 1 then
660   rows := Index(Eqs,0)-1;
661 else rows := #Eqs;
662 end if;
663  print "rows=", rows;
664 colms := Rank(P4);
665 print "colms=",colms;
666
667
668 t1 := Cputime();
```

```
print "Most time consuming step in whole";
Eqs1:=Eqs;
t2 := Cputime(t1);
print "Time for converting to lower dim poly ring=",t2;

print "Time for initialization of coeff mat";


 // Do this when i want to find the kernel
 time Matmat1 := ZeroMatrix(ExtF,rows,colms);
print "Done initialization";
t1 := Cputime();
for i in [1..rows] do
  temp_c,temp_m := CoefficientsAndMonomials(Eqs1[i]);
  count :=1;
  for j in [1..#temp_m] do
    temp_mon := Sprint(temp_m[j]);
    k := StringToInteger(Split(temp_mon,".")[2]);
    //print k;
    Matmat1[i][k] := Matmat1[i][k] + temp_c[j];
  end for;
end for;
t2 := Cputime(t1);
print "Time for Matmat1=",t2;
Matmat := Matmat1;

print "\n Calculating Kernel\n";
t12 := Cputime();


// do this when I want to know the struucture

/*
time Matmat1 := ZeroMatrix(P4,rows,colms);
print "Done initialization";
t1 := Cputime();

for i in [1..rows] do

  temp_c,temp_m := CoefficientsAndMonomials(Eqs1[i]);
  count :=1;
  for j in [1..#temp_m] do
    temp_mon := Sprint(temp_m[j]);
    kk := Split(temp_mon,"*");
    //print kk;
    if #kk gt 1 then
      k := StringToInteger(Split(kk[2],".")[2]);
      kkk := Split(kk[1],".")[2];
      kkkk := Split(kkk,"^");
      if #kkkk eq 1 then
        Append(~kkkk,"1");
```

```
      end if;
      //print kkkk;
      Matmat1[i][k] := Matmat1[i][k] + temp_c[j]*P4.
   StringToInteger(kkkk[1])^StringToInteger(kkkk[2]);
    else
      k := StringToInteger(Split(kk[1],".")[2]);
    //print k;
      Matmat1[i][k] := Matmat1[i][k] + temp_c[j];
    end if;
  end for;
end for;
t2 := Cputime(t1);
print "Time for Matmat1=",t2;



J,K := Support(pi_F1ex);
pi_F1Extst := P2!0;
for i in [1..#J] do
  pi_F1Extst  := pi_F1Extst + P2!(P2!K[i]*(chi_x1^(J[i])));
end for;
//pi_F1Extst := NormalForm(pi_F1Extst,fieldeq2);
*/

Matmat := Matmat1;
time KK := KernelMatrix(Transpose(Matmat));
t13 := Cputime(t12);
print "\n DOne computing kernel, time req=", t13, "\n";
print "\nNumber of rows in Kernel=", NumberOfRows(KK);
Model :=[ExtF!Random(q) : i in [1..NumberOfRows(KK)]]; // seq of
   constants to multiply with
KK2 := Model[1]*KK[1];


for i in [2..NumberOfRows(KK)] do
  if i gt colms then
    break;
  end if;
  KK2 := KK2 + Model[i]*KK[i];
end for;
//print "Model=",Model;
KK1 :=KK2;

Sol := [KK1[i] : i in [1..NumberOfColumns(KK1)]];
//Sol2 := [ExtF!0 : i in [1..NumberOfColumns(KK)]];
for i in [1..#Supsup] do
  if Evaluate(TT11[Index(SS11,Supsup[i])],Sol)  ne 0 then
    print "Anamonly here";
  end if;
end for;

```

```
t1:= Cputime();
c1_x := P2!0;
time restSup := SetToSequence(SequenceToSet(SS11) diff
   SequenceToSet(Supsup));
Sort(~restSup);
for j in [1..#restSup] do
  c1_x := c1_x + Evaluate(TT11[Index(SS11,restSup[j])],Sol)*
   Ext_var_pow(chi_x1,restSup[j]);//SS11[j]);
  //c1_x := c1_x + Evaluate(TT11[Index(SS11,restSup[j])],Sol)*((
   chi_x1^restSup[j]));
end for;
t2 := Cputime(t1);print "New intermediate pols created, time =",t2
    ;
t10 := Cputime();

/*
if c1_x eq 0 then
  print "No new quations found";
else print "Good New quations found";
end if;

time c1_x1 := NormalForm(c1_x,fieldeq2);
t11 := Cputime(t10);
print "time req to calc c1_x= ", t11;



//if c1_x ne P2!0 then
c1_x1_mat := Pol_mat(c1_x1);
c1_x1_bas := [c1_x1_mat[i][1] : i in [1..NumberOfRows(c1_x1_mat)
   ]];
ciph :=[];
zerociph := [0 : i in [1..#c1_x1_bas]];
for i in [1..#c1_x1_bas] do
Evaluate(c1_x1_bas[i],s);
end for;


print "\n Doing GB calculation with new eqs\n ";
t1 := Cputime();
//G1 :=GroebnerBasis(C9 cat c1_x1_bas);
G1 :=GroebnerBasis(C9 cat c1_x1_bas,4);// cat c2_x2_bas,3);

t2 := Cputime(t1);
print "new Gb time is =", t2;
G1;

*/
//the following is to recover all possible intermediate equations
countloop :=0;
eqsset :=0;
```

```
if c1_x eq 0 then
  print "c1x eq 0";
else eqsset :=1;
end if;

while (c1_x eq 0) and (eqsset eq 0) do
  print "In loop ", countloop;
  print "No new quations found";
  Model :=[ExtF!Random(q) : i in [1..NumberOfRows(KK)]];
  KK2 := Model[1]*KK[1];


  for i in [2..NumberOfRows(KK)] do
  if i gt colms then
    break;
  end if;
  KK2 := KK2 + Model[i]*KK[i];//
  end for;
  KK1 :=KK2;
  t1:= Cputime();
  c1_x := P2!0;
  time restSup := SetToSequence(SequenceToSet(SS11) diff
   SequenceToSet(Supsup));
  for j in [1..#restSup] do
    c1_x := c1_x + Evaluate(TT11[Index(SS11,restSup[j])],Sol)*
   Ext_var_pow(chi_x1,restSup[j]);//SS11[j]);
    //c1_x := c1_x + Evaluate(TT11[j],Sol)*Ext_var_pow(chi_x1,SS11
   [j]);
  end for;
  t2 := Cputime(t1);
  countloop := countloop +1;
  if countloop gt 1000 then
    print "Too many loops with no result";
    print "No new quations found";
    eqsset :=0;
    break;
  end if;
end while;
//check to see if the equations chosen are actually evaluating to
   0 or not
//Sol := [KK1[i] : i in [1..NumberOfColumns(KK1)]];
//Sol2 := [ExtF!0 : i in [1..NumberOfColumns(KK)]];
//for i in [1..#Supsup] do
//  Evaluate(Eqs4[i], Sol);
//end for;

if eqsset eq 1 then
  print "Good New quations found";
  time c1_x1 := NormalForm(c1_x,fieldeq2);
  t11 := Cputime(t10);
  print "time req to calc c1_x= ", t11;
```

```
  c1_x1_mat := Pol_mat(c1_x1);
  c1_x1_bas := [c1_x1_mat[i][1] : i in [1..n]];
  for i in [1..#c1_x1_bas] do
    Evaluate(c1_x1_bas[i],s);
  end for;

////////////////////////////////////////////////////////////////
/////// Second polynomial///////////////
/*
tmpbas :=[];
for k in [1..4] do
Model2 :=[ExtF!Random(0,1) : i in [1..NumberOfRows(KK)]];
KK4 := Model2[1]*KK[1];


for i in [2..NumberOfRows(KK)] do
  if i gt colms then
    break;
  end if;
  KK4 := KK4 + Model2[i]*KK[i];//
end for;
Sol2 := [KK4[i] : i in [1..NumberOfColumns(KK4)]];

t1:= Cputime();

c2_x := P2!0;
time restSup := SetToSequence(SequenceToSet(SS11) diff
   SequenceToSet(Supsup));

for j in [1..#restSup] do
//for j in [1..#SS11] do
  c2_x := c2_x + Evaluate(TT11[Index(SS11,restSup[j])],Sol2)*
   Ext_var_pow(chi_x1,restSup[j]);//SS11[j]);
//  c1_x := c1_x + Evaluate(TT11[j],Sol)*Ext_var_pow(chi_x1,SS11[j
   ]);
end for;
t2 := Cputime(t1);print "New intermediate pols created, time =",t2
    ;

t10 := Cputime();

if c2_x eq 0 then
  print "No new quations found";
else print "Good New quations found";
end if;

time c2_x2 := NormalForm(c2_x,fieldeq2);
//print "Normally",c1_x1;
t11 := Cputime(t10);
print "time req to calc c1_x= ", t11;

```

```
//if c1_x ne P2!0 then
c2_x2_mat := Pol_mat(c2_x2);
c2_x2_bas := [c2_x2_mat[i][1] : i in [1..n]];

tmpbas := tmpbas cat c2_x2_bas;
end for;


*/
////////////////////////////////////////////////////////////

  print "\n Doing GB calculation with new eqs\n ";
  t1 := Cputime();
  G1 :=GroebnerBasis(C9 cat c1_x1_bas);// cat c2_x2_bas,3);
  t2 := Cputime(t1);
  print "new Gb time is =", t2;
end if;
G1;
```

Appendix B

CFPKM-Source Code

This is the C code for CFPKM.

This is "api.h". This defines the global constants needed by the interface which runs the C code for evaluation of the scheme. This header is a common header for every scheme submitted to the NIST competition.

```
#ifndef api_h
#define api_h
#include "KEMheader.h"
#include <math.h>
#define CRYPTO_SECRETKEYBYTES N+SEEDSIZE
#define CRYPTO_PUBLICKEYBYTES PK_LENGTH+SEEDSIZE
#define CRYPTO_BYTES M
#define CRYPTO_CIPHERTEXTBYTES PK_LENGTH+M

#define CRYPTO_ALGNAME "CFPKM"
int crypto_kem_enc( unsigned char *ct, unsigned char *ss, const
    unsigned char *pk);
int crypto_kem_keypair( unsigned char *pk, unsigned char *sk);
int crypto_kem_dec( unsigned char *ss, const unsigned char *ct,
    const unsigned char *sk);

#endif /* api_h */
```

This is "KEMheader.h". This defines the CFPKM specific constants which are used to define the security of the parameters of the scheme. This also defines the scheme specific functions required in key-exchange.

```
#ifndef _HEAD_
#define _HEAD_
#include <stdint.h>
#include <math.h>

#define LAMBDA 256
```

```
#define SEEDSIZE 67
#define LOG2_Q 55 /* log_2 q.*/
#define N 115      /* number of variables.*/
#define B 6/* Number of bits extracted from a element.*/
#define M 116    /*the number of equations*/

#define Q 36028797018963968

#define COFSIZE 16384  /*bound on the bitsize of the coeffeicients
    of the polynomials*/
#define SECRETVAL_LENGTH 1
#define SHAREDKEYSIZE (M*B/8)
#define ERROR_LENGTH 1
#define PK_LENGTH (M*8)
#define RANGE 6
#define B_BAR  (LOG2_Q-B)
typedef struct {
    long *QD;
    long *L;
    long C;
  }Pol;


void allocatemem(Pol *f, int n,int m);
void freealloc(Pol *f, int m);
void polgen(Pol *f, int m, int n );
unsigned long long evaluate_poly(Pol unPoly, unsigned char *pValue
   , int n);
void Eval_sys(Pol *pSyst, unsigned char* pValue, int m, int n,
   unsigned long long*result);
unsigned char rounding(unsigned long long in);
void kem_rounding(unsigned char *out,  unsigned long long *in);
void kem_rec(unsigned char *key,  unsigned long long *b, unsigned
   char *c);

int crypto_kem_keypair(unsigned char *pk, unsigned char *sk);

int crypto_kem_enc(unsigned char *ct, unsigned char *ss, const
   unsigned char *pk);

int crypto_kem_dec(unsigned char *ss, const unsigned char *ct,
   const unsigned char *sk);



unsigned char kem_crossround1( unsigned long long in);
void kem_crossround2(unsigned char *out, unsigned long long *in);
void pack_sk(unsigned char *sk, unsigned char *sa, unsigned char *
   seed);
void unpack_sk(unsigned char *sa, unsigned char *seed,const
   unsigned char *sk);
```

```
void pack_pk(unsigned char *pk,unsigned long long *b1, unsigned
    char *seed);
void unpack_pk(unsigned long long *b1, unsigned char *seed,const
    unsigned char *pk);
void pack_ct(unsigned char *ct,unsigned long long *b2, unsigned
    char *c);
void unpack_ct( unsigned long long *b2, unsigned  char *c, const
    unsigned char *ct);
#endif
```

The following two sections of codes are "randombytes.h" and "rng.h". These are codes as preset by NIST for generating the seed by using a secure Pseudo Random Number (PNR) generator.

```
#ifndef RANDOMBYTES_H
#define RANDOMBYTES_H

#define _GNU_SOURCE

#include "rng.h"
#endif
```

This is "rng.h".

```
/*
  rng.h

  Created by Bassham, Lawrence E (Fed) on 8/29/17.
  Copyright    2017 Bassham, Lawrence E (Fed). All rights reserved
    .
*/

#ifndef rng_h
#define rng_h

#include <stdio.h>

#define RNG_SUCCESS      0
#define RNG_BAD_MAXLEN  -1
#define RNG_BAD_OUTBUF  -2
#define RNG_BAD_REQ_LEN -3

typedef struct {
    unsigned char   buffer[16];
    int             buffer_pos;
    unsigned long   length_remaining;
    unsigned char   key[32];
    unsigned char   ctr[16];
} AES_XOF_struct;

```

```
typedef struct {
    unsigned char   Key[32];
    unsigned char   V[16];
    int             reseed_counter;
} AES256_CTR_DRBG_struct;


void
AES256_CTR_DRBG_Update(unsigned char *provided_data,
                       unsigned char *Key,
                       unsigned char *V);

int
seedexpander_init(AES_XOF_struct *ctx,
                  unsigned char *seed,
                  unsigned char *diversifier,
                  unsigned long maxlen);

int
seedexpander(AES_XOF_struct *ctx, unsigned char *x, unsigned long
   xlen);

void
randombytes_init(unsigned char *entropy_input,
                 unsigned char *personalization_string,
                 int security_strength);

int
randombytes(unsigned char *x, unsigned long long xlen);
void   AES256_ECB(unsigned char *key, unsigned char *ctr, unsigned
    char *buffer);

#endif /* rng_h */
```

This is "kem.c". This is the implementation of the key Encapsulation mechanism of CFPKM[▶oc: nned to mention for which parameter is this].

```
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include "api.h"
#include "KEMheader.h"
#include "rng.h"
#include "randombytes.h"



void allocatemem(Pol *f, int n,int m){

```

```
  int i;
  for(i =0;i<m;i++)
  {
    f[i].QD = malloc((n*n) * sizeof(long));            /*
   allocating memory for the coefficients of each polynomial to be
    stored in*/
    f[i].L = malloc(n * sizeof(long));

  }


}


void freealloc(Pol *f, int m)
{
  int i;
  for(i=0;i<m;i++)               /* freeing the allocated memory*/
  {
    free(f[i].QD);
    free(f[i].L);

  }

}


void polgen(Pol *f, int m, int n )
{                                                                  /*
   generates a system of m polynomials over n variables */
  int i,l;

  long *out=malloc(sizeof( long));

  for(i=0; i< m; i++)
  {


    long cofval[(N*(N+1)/2) + N+1];


    for (l=0;l< ((N*(N+1)/2) + N+1);l++)
    {
      randombytes((unsigned char*)(&out), 4);
      cofval[l]=((long)out)%(COFSIZE);
    }

    int j,k,count=0;

    for(j=0; j<n; j++)
    {
```

```
      for(k=0; k<n; k++)
      {

        if(k > j)
          f[i].QD[(k*n+j)] = 0;
        else
          {
          f[i].QD[(k*n+j)] = cofval[count]%(COFSIZE);
          count++;
          }
      }
    }

    for(j=0; j<n; j++)
    {
      f[i].L[j] = cofval[count]%(COFSIZE) ;
      count++;
    }

    f[i].C = cofval[count]%(COFSIZE) ;


  }

}




unsigned long long evaluate_poly(Pol unPoly,  unsigned char *
   pValue, int n)
{
  int i, j;
  unsigned long long result1 = 0, result2 = 0;
              /* evaluates f over a value, like f(sa)*/
  unsigned long long tabResult1[n];
/*for quad */
  for(j=0; j<n; j++)
  {
    tabResult1[j] =0;
    for(i=0; i<n; i++)
    {
      tabResult1[j] = tabResult1[j]+ ((unsigned long)pValue[i] *
   unPoly.QD[i*n + j]) ;



    }
    result1 = (result1 + tabResult1[j] * (unsigned long)pValue[j])
    ;
```

```
  }
/*for linear*/
  for(i=0; i<n; i++)
  {

    result2 = (result2 + unPoly.L[i] * (unsigned long)pValue[i]) ;
  }

  result1 = (result1 + result2 + unPoly.C);

  return result1;
}

void Eval_sys(Pol *pSyst, unsigned char* pValue, int m, int n,
    unsigned long long *result)
{
  int i;
                              /*evaluates a system of polynomials
    over a provided value, calls the evaluate_poly function for
    each polynomial*/
  for(i=0; i<M; i++)
    result[i] = evaluate_poly(pSyst[i], pValue, N);

}




unsigned char kem_crossround1( unsigned long long in){
  unsigned char out;
  unsigned long long rem = in >> (B_BAR-1);                  /*
    CrossRound function to give the CrossRound bit of a value*/
  out =(unsigned char) (rem%2);
  return out;
}

unsigned char rounding(unsigned long long in)
{
  unsigned char out;
  unsigned long long rem =( in + (2^(B_BAR-1)));             /*
    Rounding function to give the rounded value*/
  unsigned long long rem2 = (rem % Q);
  out = (unsigned char)((rem2 >> B_BAR));

  return out;

}

void kem_crossround2(unsigned char *out,  unsigned  long long *in)
    {
  int i;
```

```
154
        /*CrossRound function over a vector*/
155   for (i = 0; i < M; i++) {
156     unsigned long long rem = in[i] >> (B_BAR-1);
157     out[i] = (unsigned char)(rem%2);
158   }
159
160 }
161
162
163 void kem_rounding(unsigned char *out,  unsigned long long *in) {
164   int i;
165   for (i=0 ; i < M ; i++)                    /*Rounding function
       over a vector*/
166   {
167     unsigned long long rem = (in[i] + (2^(B_BAR-1)));
168     unsigned long long rem2 = (rem % Q);
169     out[i] = (unsigned char)((rem2 >> B_BAR));
170   }
171
172 }
173
174 void kem_rec(unsigned char *key,  unsigned long long *w, unsigned
       char *c){
175   int i;
176   unsigned long long w1,w2;
177   unsigned char hint;
178   for (i =0; i < M;i++){                          /*Red function
       from the article*/
179     int flag=0;
180     hint= kem_crossround1(w[i]);
181     if (hint==c[i])
182     {
183       key[i] = rounding(w[i]);
184       flag=1;
185     }
186     if (flag==0)
187     {
188       w1 = (w[i] + (2^(B_BAR-2))-1) ;
189       hint= kem_crossround1(w1);
190       if (hint==c[i]){
191         key[i] = rounding(w1);
192       }
193       else{
194         w2 =(w[i] - (2^(B_BAR-2))+1) ;
195         hint= kem_crossround1(w2);
196         if (hint==c[i]){
197           key[i] = rounding(w2);
198         }
199         else key[i]=0;
200       }
```

```
    }
  }

}

void pack_sk(unsigned char *sk, unsigned char *sa, unsigned char *
    seed){

  int i;                                    /* makes SK=(seed||sa)*/
  for(i=0;i<  SEEDSIZE;i++)
    {sk[i]=seed[i];}
  for(i=0;i < N;i++)
    sk[SEEDSIZE+i]=sa[i];
}
void unpack_sk(unsigned char *sa, unsigned char *seed, const
    unsigned char *sk){

  int i;
  for(i=0;i< SEEDSIZE;i++)                          /*unpacks SK to
    give out seed and sa*/
    {seed[i]=sk[i];}
  for(i=0;i < N;i++)
    sa[i]=sk[SEEDSIZE+i];


}
void pack_pk(unsigned char *pk,unsigned  long long *b1, unsigned
    char *seed){


  int i,j;
  for(i=0 ;i <SEEDSIZE;i++)
    {pk[i]=seed[i];}
  unsigned char temp;
  unsigned char mask=255;                         /* makes PK=(seed
    ||b1)*/
  for(i =0;i<M;i++)
    {for(j=7;j>-1;j--)
      {temp=(b1[i] & mask);
      b1[i]=b1[i]>>8;
      pk[SEEDSIZE+i*8+j]=temp;
    }
  }

}
void unpack_pk(unsigned long long *b1, unsigned char *seed, const
    unsigned char *pk){
  int i,j;
  for(i=0;i<SEEDSIZE;i++)
    seed[i]=pk[i];
  unsigned char temp;
```

```
246   for(i=0;i<M;i++)
247     b1[i]=0;
248   for(i=0;i<M;i++)
249     {                                 /*unpacks PK to give out seed
      and the public vector b1*/
250       for(j=0;j<7;j++)
251         {
252         temp = pk[i*8+j+SEEDSIZE];
253         b1[i]=b1[i]+temp;
254         b1[i]=b1[i]<<8;
255         }
256       b1[i]=b1[i]+pk[i*8+7+SEEDSIZE];
257     }
258
259
260 }
261 void pack_ct(unsigned char *ct, unsigned long long *b2,unsigned
      char *c){
262
263
264   int  i,j;
265   for (i=0;i < M;i++)
266     ct[i]=c[i];
267                                       /*makes ct=(c||b2)*/
268   unsigned char temp;
269   unsigned char mask=255;
270   for(i =0;i<M;i++)
271     {for(j=7;j>-1;j--)
272       {temp=(unsigned char)(b2[i] & mask);
273       b2[i]=b2[i]>>8;
274       ct[M+i*8+j]=temp;
275     }
276   }
277
278
279
280
281 }
282 void unpack_ct(unsigned long long *b2,unsigned  char *c, const
      unsigned char *ct){
283
284   int  i,j;
285   for (i=0;i < M;i++)
286     c[i]=ct[i];
287
288   unsigned char temp;
289   for(i=0;i<M;i++)                          /*unpacks ct to give out
      the hint vector c and b2*/
290     b2[i]=0;
291   for(i=0;i<M;i++)
292     {
```

```
      for(j=0;j<7;j++)
        {
        temp = ct[i*8+j+M];
        b2[i]=b2[i]+temp;
        b2[i]=b2[i]<<8;
        }
      b2[i]=b2[i]+ct[i*8+7+M];
    }

}


int crypto_kem_keypair(unsigned char *pk, unsigned char *sk){

  unsigned char *seed=malloc(SEEDSIZE*sizeof(unsigned char));if (
   seed==NULL) {printf("EXIT");return 0;}
  randombytes(seed,SEEDSIZE);


  Pol *f1 = malloc(M * sizeof(Pol));
  allocatemem(f1,N,M);

  randombytes_init(seed,NULL,256);
  polgen(f1,M,N);

  int i;

  unsigned char *sa=malloc(N*sizeof(unsigned char));if (sa==NULL)
   {printf("EXIT");return 0;}


  randombytes(sa,N*SECRETVAL_LENGTH);

  unsigned char *e1=malloc(M*sizeof(unsigned char));if (e1==NULL)
   {printf("EXIT");return 0;}
  randombytes(e1,M*ERROR_LENGTH);


  for(i=0;i < N;i++)
    sa[i]=(unsigned char)((sa[i])%RANGE);


  for(i=0;i < M;i++)
    {e1[i]=(unsigned char)((e1[i])%RANGE);  }


  unsigned long long *b1=malloc(M*sizeof(unsigned long long));if (
   b1==NULL) {printf("EXIT");return 0;}
  Eval_sys(f1,sa,M,N,b1);
  for (i =0;i <M ;i++)
  {
```

```
    b1[i] = (b1[i] + e1[i]) ;


  }

  pack_sk(sk,sa,seed);

  pack_pk(pk,b1,seed);


  return 0;
}

int crypto_kem_enc(unsigned char *ct, unsigned char *ss, const
    unsigned char *pk){
  int i;
  unsigned long long *b1=malloc(M*sizeof(unsigned long long));
  unsigned char *seed=malloc(SEEDSIZE*sizeof(unsigned char));
  unpack_pk(b1, seed, pk);


  Pol *f2 = malloc(M*sizeof(Pol));
  allocatemem(f2,N,M);

  randombytes_init(seed,NULL,256);
  polgen(f2,M,N);



  unsigned char *seed1=malloc(SEEDSIZE*sizeof(unsigned char));
  randombytes(seed1,SEEDSIZE);

  randombytes_init(seed1,NULL,256);
  unsigned char *sb=malloc(N*sizeof(unsigned char));
  unsigned char *e2=malloc(M*sizeof(unsigned char));if (e2==NULL)
    {printf("EXIT");return 0;}
  unsigned char *e3=malloc(M*sizeof(unsigned char));if (e3==NULL)
    {printf("EXIT");return 0;}

  randombytes(sb, N*SECRETVAL_LENGTH);

  randombytes(e2,M*ERROR_LENGTH);
  randombytes(e3,M*ERROR_LENGTH);

  for(i=0;i < N;i++)
    {sb[i]=(unsigned char)((sb[i])%RANGE);}


  for(i=0;i < M;i++)
    {e2[i]=(unsigned char)((e2[i])%RANGE);
    e3[i]=(unsigned char)((e3[i])%RANGE); }
```

```
388
389
390
391   unsigned long long *b2=malloc(M*sizeof(unsigned long long));if (
       b2==NULL) {printf("EXIT");return 0;}
392   unsigned long long *b3=malloc(M*sizeof(unsigned long long));if (
       b3==NULL) {printf("EXIT");return 0;}
393
394   Eval_sys(f2,sb,M,N,b2);
395
396   for  (i =0;i<M;i++){
397     b3[i] = (b2[i]*b1[i] + e3[i]);
398     b2[i] = (b2[i] + e2[i]);
399
400         }
401
402   kem_rounding(ss, b3);
403
404   unsigned char *c=malloc(M*sizeof(unsigned char));
405   kem_crossround2(c, b3);
406   pack_ct(ct, b2, c);
407
408   return 0;
409 }
410
411
412 int crypto_kem_dec(unsigned char *ss, const unsigned char *ct,
       const unsigned char *sk){
413   int i;
414   unsigned char *sa=malloc(N*sizeof(unsigned char));
415   unsigned char *seed=malloc(SEEDSIZE*sizeof(unsigned char));
416   unpack_sk(sa,seed,sk);
417   unsigned long long *b2=malloc(M*sizeof(unsigned long long));
418   unsigned char *c=malloc(M*sizeof(unsigned char));
419
420   unpack_ct(b2,c,ct);
421   Pol *f = (Pol*)malloc(M*sizeof(Pol));
422   allocatemem(f,N,M);
423
424   randombytes_init(seed,NULL,256);
425   polgen(f,M,N);
426
427   unsigned long long *w = malloc(M*sizeof(unsigned long long));
428   Eval_sys(f,sa,M,N,w);
429   for (i=0;i < M;i++)
430     {
431       w[i]=(w[i]*b2[i]) ;}
432
433
434
435
```

```
  kem_rec(ss, w, c);

  return 0;
}
```

```

/*
//  PQCgenKAT_kem.c
//
//  Created by Bassham, Lawrence E (Fed) on 8/29/17.
//  Copyright    2017 Bassham, Lawrence E (Fed). All rights reserved.
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "rng.h"
#include "api.h"
#include "KEMheader.h"

#define MAX_MARKER_LEN    50
#define KAT_SUCCESS          0
#define KAT_FILE_OPEN_ERROR -1
#define KAT_DATA_ERROR      -3
#define KAT_CRYPTO_FAILURE  -4

int   FindMarker(FILE *infile, const char *marker);
int   ReadHex(FILE *infile, unsigned char *A, int Length, char *str);
void  fprintBstr(FILE *fp, char *S, unsigned char *A, unsigned long long L);

int
main()
{
    char                fn_req[32], fn_rsp[32];
    FILE                *fp_req, *fp_rsp;
    unsigned char       seed[48];
    unsigned char       entropy_input[48];
    unsigned char       ct[CRYPTO_CIPHERTEXTBYTES], ss[CRYPTO_BYTES], ss1[CRYPTO_BYTES];
    int                 count;
    int                 done;
    unsigned char       pk[CRYPTO_PUBLICKEYBYTES], sk[CRYPTO_SECRETKEYBYTES];
    int                 ret_val;

    /* Create the REQUEST file*/
    sprintf(fn_req, "PQCkemKAT_%d.req", CRYPTO_SECRETKEYBYTES);
    if ( (fp_req = fopen(fn_req, "w")) == NULL ) {
```

```
        printf("Couldn't open <%s> for write\n", fn_req);
        return KAT_FILE_OPEN_ERROR;
    }
    sprintf(fn_rsp, "PQCkemKAT_%d.rsp", CRYPTO_SECRETKEYBYTES);
    if ( (fp_rsp = fopen(fn_rsp, "w")) == NULL ) {
        printf("Couldn't open <%s> for write\n", fn_rsp);
        return KAT_FILE_OPEN_ERROR;
    }
    int i;
    for (i=0; i<48; i++)
        entropy_input[i] = i;

    randombytes_init(entropy_input, NULL, 256);
    for (i=0; i<100; i++) {
        fprintf(fp_req, "count = %d\n", i);
        randombytes(seed, 48);
        fprintBstr(fp_req, "seed = ", seed, 48);
        fprintf(fp_req, "pk =\n");
        fprintf(fp_req, "sk =\n");
        fprintf(fp_req, "ct =\n");
        fprintf(fp_req, "ss =\n\n");
    }
    fclose(fp_req);

    /*Create the RESPONSE file based on what's in the REQUEST file
    */
    if ( (fp_req = fopen(fn_req, "r")) == NULL ) {
        printf("Couldn't open <%s> for read\n", fn_req);
        return KAT_FILE_OPEN_ERROR;
    }

    fprintf(fp_rsp, "# %s\n\n", CRYPTO_ALGNAME);
    done = 0;
    do {
        if ( FindMarker(fp_req, "count = ") )
            fscanf(fp_req, "%d", &count);
        else {
            done = 1;
            break;
        }
        fprintf(fp_rsp, "count = %d\n", count);

        if ( !ReadHex(fp_req, seed, 48, "seed = ") ) {
            printf("ERROR: unable to read 'seed' from <%s>\n",
    fn_req);
            return KAT_DATA_ERROR;
        }
        fprintBstr(fp_rsp, "seed = ", seed, 48);

        randombytes_init(seed, NULL, 256);

```

```
        /* Generate the public/private keypair*/
        if ( (ret_val = crypto_kem_keypair(pk, sk)) != 0) {
            printf("crypto_kem_keypair returned <%d>\n", ret_val);
            return KAT_CRYPTO_FAILURE;
        }
        fprintBstr(fp_rsp, "pk = ", pk, CRYPTO_PUBLICKEYBYTES);
        fprintBstr(fp_rsp, "sk = ", sk, CRYPTO_SECRETKEYBYTES);

        if ( (ret_val = crypto_kem_enc(ct, ss, pk)) != 0) {
            printf("crypto_kem_enc returned <%d>\n", ret_val);
            return KAT_CRYPTO_FAILURE;
        }
        fprintBstr(fp_rsp, "ct = ", ct, CRYPTO_CIPHERTEXTBYTES);
        fprintBstr(fp_rsp, "ss = ", ss, CRYPTO_BYTES);

        fprintf(fp_rsp, "\n");

        if ( (ret_val = crypto_kem_dec(ss1, ct, sk)) != 0) {
            printf("crypto_kem_dec returned <%d>\n", ret_val);
            return KAT_CRYPTO_FAILURE;
        }

        if ( memcmp(ss, ss1, CRYPTO_BYTES) ) {
            printf("crypto_kem_dec returned bad 'ss' value\n");
            return KAT_CRYPTO_FAILURE;
        }

    } while ( !done );

    fclose(fp_req);
    fclose(fp_rsp);

    return KAT_SUCCESS;
}



/*
// ALLOW TO READ HEXADECIMAL ENTRY (KEYS, DATA, TEXT, etc.)
//
//
// ALLOW TO READ HEXADECIMAL ENTRY (KEYS, DATA, TEXT, etc.)
*/
int
FindMarker(FILE *infile, const char *marker)
{
  char  line[MAX_MARKER_LEN];
  int   i, len;
  int curr_line;

  len = (int)strlen(marker);
```

```
142   if ( len > MAX_MARKER_LEN-1 )
143     len = MAX_MARKER_LEN-1;
144 
145   for ( i=0; i<len; i++ )
146     {
147       curr_line = fgetc(infile);
148       line[i] = curr_line;
149       if (curr_line == EOF )
150         return 0;
151     }
152   line[len] = '\0';
153 
154   while ( 1 ) {
155     if ( !strncmp(line, marker, len) )
156       return 1;
157 
158     for ( i=0; i<len-1; i++ )
159       line[i] = line[i+1];
160     curr_line = fgetc(infile);
161     line[len-1] = curr_line;
162     if (curr_line == EOF )
163         return 0;
164     line[len] = '\0';
165   }
166 
167   /* shouldn't get here*/
168   return 0;
169 }
170 
171 /*
172 // ALLOW TO READ HEXADECIMAL ENTRY (KEYS, DATA, TEXT, etc.)
173 */
174 int
175 ReadHex(FILE *infile, unsigned char *A, int Length, char *str)
176 {
177   int     i, ch, started;
178   unsigned char ich;
179 
180   if ( Length == 0 ) {
181     A[0] = 0x00;
182     return 1;
183   }
184   memset(A, 0x00, Length);
185   started = 0;
186   if ( FindMarker(infile, str) )
187     while ( (ch = fgetc(infile)) != EOF ) {
188       if ( !isxdigit(ch) ) {
189         if ( !started ) {
190           if ( ch == '\n' )
191             break;
192           else
```

```
            continue;
        }
        else
          break;
      }
      started = 1;
      if ( (ch >= '0') && (ch <= '9') )
        ich = ch - '0';
      else if ( (ch >= 'A') && (ch <= 'F') )
        ich = ch - 'A' + 10;
      else if ( (ch >= 'a') && (ch <= 'f') )
        ich = ch - 'a' + 10;
            else /* shouldn't ever get here*/
                ich = 0;

      for ( i=0; i<Length-1; i++ )
        A[i] = (A[i] << 4) | (A[i+1] >> 4);
      A[Length-1] = (A[Length-1] << 4) | ich;
    }
  else
    return 0;

  return 1;
}

void
fprintBstr(FILE *fp, char *S, unsigned char *A, unsigned long long
    L)
{
  unsigned long long  i;

  fprintf(fp, "%s", S);

  for ( i=0; i<L; i++ )
    fprintf(fp, "%02X", A[i]);

  if ( L == 0 )
    fprintf(fp, "00");

  fprintf(fp, "\n");
}
```

```
/*
  rng.c

  Created by Bassham, Lawrence E (Fed) on 8/29/17.
  Copyright    2017 Bassham, Lawrence E (Fed). All rights reserved
    .
*/
#include <stdio.h>
#include <stdlib.h>
```

```c
#include "string.h"
#include "rng.h"

#include <openssl/conf.h>
#include <openssl/evp.h>
#include <openssl/err.h>

AES256_CTR_DRBG_struct  DRBG_ctx;


/*
 seedexpander_init()
 ctx            - stores the current state of an instance of the seed expander
 seed           - a 32 byte random value
 diversifier    - an 8 byte diversifier
 maxlen         - maximum number of bytes (less than 2**32) generated under this seed and diversifier
 */
int
seedexpander_init(AES_XOF_struct *ctx,
                  unsigned char *seed,
                  unsigned char *diversifier,
                  unsigned long maxlen)
{
    if ( maxlen >= 0x100000000 )
        return RNG_BAD_MAXLEN;

    ctx->length_remaining = maxlen;

    memcpy(ctx->key, seed, 32);

    memcpy(ctx->ctr, diversifier, 8);
    ctx->ctr[11] = maxlen % 256;
    maxlen >>= 8;
    ctx->ctr[10] = maxlen % 256;
    maxlen >>= 8;
    ctx->ctr[9] = maxlen % 256;
    maxlen >>= 8;
    ctx->ctr[8] = maxlen % 256;
    memset(ctx->ctr+12, 0x00, 4);

    ctx->buffer_pos = 16;
    memset(ctx->buffer, 0x00, 16);

    return RNG_SUCCESS;
}

/*
 seedexpander()
    ctx  - stores the current state of an instance of the seed
```

```
   expander
    x    - returns the XOF data
    xlen - number of bytes to return
 */
int
seedexpander(AES_XOF_struct *ctx, unsigned char *x, unsigned long
   xlen)
{
    unsigned long   offset;

    if ( x == NULL )
        return RNG_BAD_OUTBUF;
    if ( xlen >= ctx->length_remaining )
        return RNG_BAD_REQ_LEN;

    ctx->length_remaining -= xlen;

    offset = 0;
    while ( xlen > 0 ) {
        /* XXX I add (unsigned) to remove a warning during the
   compilation XXX */
        if ( xlen <= (unsigned)(16-ctx->buffer_pos) ) { /* buffer
   has what we need*/
            memcpy(x+offset, ctx->buffer+ctx->buffer_pos, xlen);
            ctx->buffer_pos += xlen;

            return RNG_SUCCESS;
        }

        /* take what's in the buffer*/
        memcpy(x+offset, ctx->buffer+ctx->buffer_pos, 16-ctx->
   buffer_pos);
        xlen -= 16-ctx->buffer_pos;
        offset += 16-ctx->buffer_pos;

        AES256_ECB(ctx->key, ctx->ctr, ctx->buffer);
        ctx->buffer_pos = 0;

       int i;
        for ( i=15; i>=12; i--) {
            if ( ctx->ctr[i] == 0xff )
                ctx->ctr[i] = 0x00;
            else {
                ctx->ctr[i]++;
                break;
            }
        }

    }

    return RNG_SUCCESS;
```

```
}


void handleErrors(void)
{
    ERR_print_errors_fp(stderr);
    abort();
}

/* Use whatever AES implementation you have. This uses AES from openSSL library
    key - 256-bit AES key
    ctr - a 128-bit plaintext value
   buffer - a 128-bit ciphertext value*/
void
AES256_ECB(unsigned char *key, unsigned char *ctr, unsigned char *buffer)
{
    EVP_CIPHER_CTX *ctx;

    int len;

    /* XXX I put in commentary to remove a warning during the compilation XXX */
/*    int ciphertext_len;*/

    /* Create and initialise the context */
    if(!(ctx = EVP_CIPHER_CTX_new())) handleErrors();

    if(1 != EVP_EncryptInit_ex(ctx, EVP_aes_256_ecb(), NULL, key, NULL))
        handleErrors();

    if(1 != EVP_EncryptUpdate(ctx, buffer, &len, ctr, 16))
        handleErrors();
    /* XXX I put in commentary to remove a warning during the compilation XXX */
/*    ciphertext_len = len;*/

    /* Clean up */
    EVP_CIPHER_CTX_free(ctx);
}

void
randombytes_init(unsigned char *entropy_input,
                 unsigned char *personalization_string,
                 int security_strength)
{
    unsigned char   seed_material[48];

    memcpy(seed_material, entropy_input, 48);
```

```
150     if (personalization_string)
151         {int i;
152         for (i=0; i<48; i++)
153             seed_material[i] ^= personalization_string[i];}
154     memset(DRBG_ctx.Key, 0x00, 32);
155     memset(DRBG_ctx.V, 0x00, 16);
156     AES256_CTR_DRBG_Update(seed_material, DRBG_ctx.Key, DRBG_ctx.V
        );
157     DRBG_ctx.reseed_counter = 1;
158 }
159
160 int
161 randombytes(unsigned char *x, unsigned long long xlen)
162 {
163     unsigned char   block[16];
164     int             i = 0;
165
166     while ( xlen > 0 ) {
167         /*increment V*/
168         int j;
169         for (j=15; j>=0; j--) {
170             if ( DRBG_ctx.V[j] == 0xff )
171                 DRBG_ctx.V[j] = 0x00;
172             else {
173                 DRBG_ctx.V[j]++;
174                 break;
175             }
176         }
177         AES256_ECB(DRBG_ctx.Key, DRBG_ctx.V, block);
178         if ( xlen > 15 ) {
179             memcpy(x+i, block, 16);
180             i += 16;
181             xlen -= 16;
182         }
183         else {
184             memcpy(x+i, block, xlen);
185             xlen = 0;
186         }
187     }
188     AES256_CTR_DRBG_Update(NULL, DRBG_ctx.Key, DRBG_ctx.V);
189     DRBG_ctx.reseed_counter++;
190
191     return RNG_SUCCESS;
192 }
193
194 void
195 AES256_CTR_DRBG_Update(unsigned char *provided_data,
196                        unsigned char *Key,
197                        unsigned char *V)
198 {
199     unsigned char   temp[48];
```

```
    int i,j;
    for ( i=0; i<3; i++) {
        /*increment V*/
        for (j=15; j>=0; j--) {
            if ( V[j] == 0xff )
                V[j] = 0x00;
            else {
                V[j]++;
                break;
            }
        }

        AES256_ECB(Key, V, temp+16*i);
    }
    if ( provided_data != NULL )
        {int i;
        for (i=0; i<48; i++)
            temp[i] ^= provided_data[i];}
    memcpy(Key, temp, 32);
    memcpy(V, temp+32, 16);
}
```

Appendix C

A small example to compute the matrix $\alpha_m(\mathbf{x})$

Let us consider the case of $\mathbb{F}_2$, with $n = 3$. Let us represent a polynomial ring over $\mathbb{F}_2$ as $\mathbb{F}_2[x_1, x_2, x_3]$ over the variables $\mathbf{x} = (x_1, x_2, x_3)$. So the extension field is defined by $\mathbb{F}_{2^3}$. Let ω be algebraic over $\mathbb{F}_2$. Thus we can define $\kappa = z^3 + z + 1$ as the irreducible polynomial of ω over $\mathbb{F}_2$. Now consider that the matrix $A\mathbf{x}$ gives a column vector of three polynomials, say $g_1, g_2, g_3 \in \mathbb{F}_2[\mathbf{x}]$. Hence

$$\varphi(A\mathbf{x}) = g_1 + \omega \cdot g_2 + \omega^2 \cdot g_3 \in \mathbb{F}_{2^3}[\mathbf{x}]$$

We now show the way to compute $\alpha_m(\mathbf{x}) \in$. We compute

$$\omega \cdot \varphi(A\mathbf{x}) = g_3 + \omega \cdot (g_1 + g_3) + \omega^2 \cdot g_2$$

$$\omega^2 \cdot \varphi(A\mathbf{x}) = g_2 + \omega \cdot (g_2 + g_3) + \omega^2 \cdot (g_1 + g_3)$$

Hence the matrix that represents multiplication by $\varphi(A\mathbf{x})$ with $\mathbf{x}$ is given by

$$\alpha_m(\mathbf{x}) = \begin{bmatrix} g_1 & g_2 & g_3 \\ g3 & (g_1 + g_3) & g_2 \\ g_2 & (g_2 + g_3) & (g_1 + g_3) \end{bmatrix}$$

Appendix D

Proofs from Section 4.2.5

Proposition 4.2.4. *Let $F_1, F_2 \in \mathbb{F}_q^n[x_1, \ldots, x_n]$ represent the central map polynomials of $\texttt{EFC}_q$ and $T \in \mathbb{F}_q^{2n \times 2n}$ be the linear transformation that composes with the central map $F \in \mathbb{F}_q^{2n}[x_1, \ldots, x_n]$ to form the public-key of $\texttt{EFC}$. Suppose there is a embedded forgetting map $\phi_a : \mathbb{F}_q^{2n} \mapsto \mathbb{F}_q^{2n-a} \hookrightarrow \mathbb{F}_q^{2n}$. Then for the public-keys of $\texttt{EFC}^-$, there is an equivalent representation of the linear transformation $\Phi_a \circ T$ using two distinct embedded forgetting maps $\phi_{a_1} : \mathbb{F}_q^n \mapsto \mathbb{F}_q^{n-a_1} \hookrightarrow \mathbb{F}_q^n$ and $\phi_{a_2} : \mathbb{F}_q^n \mapsto \mathbb{F}_q^{n-a_2} \hookrightarrow \mathbb{F}_q^n$ such that $a_1 + a_2 = a$ and ϕ_{a_1} acts in composition with F_1 while ϕ_{a_2} composes with F_2 of the central map, where $\circ$ is the composition map.*

Proof. The composition $E \circ T$ represents a $2n \times 2n$ matrix. This matrix has a co-rank of a. Now the rows of the this composition matrix which on composition with the central map gives out zero polynomial can be replaced by zero rows.

Now consider the first n rows of $E \circ T \circ F$. We have supposed that out of these n rows a_1 rows are zero rows. This can be represented as a composition of a $n \times n$ matrix E_1, which has the same exact a_1 zero rows, with F_1 the central map giving out the same equations as the first n rows of $E \circ T \circ F$.

Similarly take the last n rows of $E \circ T \circ F$. We can have another $n \times n$ matrix E_2 with a_2 zero rows which on composition with F_2 result in the same last n rows of $E \circ T \circ F$. Now if E forgetting map removed equations from the end of the list of public keys, a composition of a simple permutation map along with this new defined way of representation gives us the original set of $\mathcal{P}^0$. □ □

Lemma 4.2.5. *Let $\Phi_a \in \mathbb{F}_q^{2n \times 2n}$ be a linear transformation of co-rank 'a'. Also let $T \in \mathbb{F}_q^{2n \times 2n}$ be a linear transformation that composes with the central map polynomials $(F_1, F_2) \in \mathbb{F}_q^{2n}[x_1, \ldots, x_n]$. Using Proposition 4.2.4, consider there exists equivalent forgetting maps, $\Phi_{a_1} \in \mathbb{F}_q^{n \times n}$ and $\Phi_{a_2} \in \mathbb{F}_q^{n \times n}$. Also consider, the linear transformation $T \in \mathbb{F}_q^{2n \times 2n}$ to be the identity matrix. There exist a non-singular linear transformation $U \in \mathbb{F}_q^{2n \times 2n}$ and polynomials $\pi_1, \pi_2 \in \mathbb{F}_{q^n}[X]$ of degrees q^{a_1} and q^{a_2} respectively, such that $a_1 + a_2 = a$ and $\Phi_a \circ T = \Phi_a \circ I_{2n} = U \circ (\varphi^{-1}, \varphi^{-1}) \circ (\pi_1, \pi_2) \circ (\varphi, \varphi)$, where I_{2n} is the identity matrix, $\varphi : \mathbb{F}_q^n \mapsto \mathbb{F}_{q^n}$ and the composition function $\circ$ works component wise.*

Proof. From the previous proposition 4.2.4, the linear transformation $E \circ I_{2n}$ can be considered as collection of two separate forgetting maps each acting on the two sets of first n and last n polynomials, F_1 and F_2 respectively. Suppose we have a_1 polynomials removed from F_1 and a_2 removed from F_2. So, we have $a = a_1 + a_2$. Let $V_1 \in \mathbb{F}_{q^n}$ be the kernel of $E_1 \circ I_n$ and similarly $V_2 \in \mathbb{F}_{q^n}$ be the kernel of $E_2 \circ I_n$. Let π_1 be the minimal polynomial of the algebraic set V_1 and π_2 be the minimal polynomial for V_2. Now removing a_1 polynomials implies that nullity of V_1 is q^{a_1} and similarly $|V_2| = q^{a_2}$. Thus π_1 and π_2 have degrees q^{a_1} and q^{a_2} respectively and are of the form

$$\pi_1 = \sum_{i=0}^{a_1} c_i X^{q^i} \ , \ \pi_2 = \sum_{i=0}^{a_2} c_i' X^{q^i}$$

where $c_i, c_i' \in \mathbb{F}_{q^n}$ Taking the same approach as Vates and Smith-Tone (Lemma 1) [VST17], we argue that there exists linear transformations U_1 and U_2 such that

$$E_1 \circ I = U_1 \circ \varphi^{-1} \circ \pi_1 \circ \varphi \, , \ E_2 \circ I = U_2 \circ \varphi^{-1} \circ \pi_2 \circ \varphi \tag{D.1}$$

Using (D.1) , we have

$$E \circ I_{2n} = \begin{bmatrix} E_1 \circ I_n \\ E_2 \circ I_n \end{bmatrix} = \begin{bmatrix} U_1 \circ \varphi^{-1} \circ \pi_1 \circ \varphi \\ U_2 \circ \varphi^{-1} \circ \pi_2 \circ \varphi \end{bmatrix} = \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix} \cdot \begin{bmatrix} \varphi^{-1} \circ \pi_1 \circ \varphi \\ \varphi^{-1} \circ \pi_2 \circ \varphi \end{bmatrix}$$

This above matrix representation can be also written as

$$E \circ I_{2n} = U \circ (\varphi^{-1}, \varphi^{-1}) \circ (\pi_1, \pi_2) \circ (\varphi, \varphi)$$

□ □

Appendix E

Some Additional Intermediate Equations

In each case, let $\mathcal{F}_1$ and $\mathcal{F}_2$ be the public keys represented over the extension field and $\mathcal{D}_1$ and $\mathcal{D}_2$ be the corresponding ciphertext represented over the extension field on the evaluation of $\mathcal{F}_1$ and $\mathcal{F}_2$. Let $\mathcal{H}_1 = \mathcal{F}_1 - \mathcal{D}_1$ and $\mathcal{H}_2 = \mathcal{F}_2 - \mathcal{D}_2$.

1. For $\texttt{EFC}_2^-(1)$, number of quadratic polynomials observed at Step degree $3 = 4n$. We have been able to theoretically find the following 3 extension field equations which represent $3n$ equations over the base field.

    1. $(\alpha^2(\chi)\chi + \alpha(\chi))\mathcal{H}_2 + \beta(\chi)\mathcal{H}_1 = 0$

    2. $\beta^2(\chi)\mathcal{H}_1 + \alpha^2(\chi)\mathcal{H}_2^2 + \alpha(\chi)\beta(\chi)\mathcal{H}_2 = 0$

    3. $\alpha^4(\chi)\chi^2\mathcal{H}_2^2 + \beta^2(\chi)\mathcal{H}_1^2 + \alpha^2(\chi)\mathcal{H}_2^2 = 0$

2. For $\texttt{EFC}_2^{F-}(1)$, No quadratic equations are observed at any intermediate equations, but found some intermediate equations though theoretical manner, and then used those equations, found degree of regularity decrease to 3 from 4.

    1. $\beta^2(\chi)F_1 + \alpha^2(\chi)F_2^2 + \alpha(\chi)\beta(\chi)F_2 = 0$

Additionally, we found the following combinations of the public keys and their Frobenius powers for some instances of EFC.

1. For $\texttt{EFC}_2^-(2)$ with $a = (2, 0)$ we observe $3n$ quadratic polynomials at Step degree 3, while the following combination (referring to n of those) can be

easily recovered theoretically,

1. $\beta^2(\chi)F_1 + \alpha^4(\chi)\chi^2 F_2^2 + \alpha^2 F_2^2 + \alpha(\chi)\beta(\chi)F_2 = 0$

2. For $\texttt{EFC}_2^-(2)$ with $a = (1, 1)$, the number of quadratic polynomials observed at Step degree 3 is $3n$ and at least n of those can be given by the following combination

change the F1 and F2 to H1 and H2

   1. $(\beta^2(\chi)F_1 + \alpha^2(\chi)F_2)\chi + \alpha(\chi)F_2 + \beta(\chi)F_1 = 0$

3. For $\texttt{EFC}_3^-(1)$ with $a = (1, 0)$, the number of quadratic polynomials observed at Step degree 3 is $3n$ and at least n of those can be given by the following combination

   1. $\big(\beta^3(\chi)F_1 - \alpha^3(\chi)F_2^3 - \alpha(\chi)\beta^2(\chi)F_2\big)\chi - M_2\alpha(\chi)\beta(\chi)F_2 = 0$

Résumé

La résolution de systèmes polynomiaux est l'un des problèmes les plus anciens et des plus importants en Calcul Formel et a de nombreuses applications. C'est un problème intrinsèquement difficile avec une complexité, en générale, au moins exponentielle en le nombre de variables. Dans cette thèse, nous nous concentrons sur des schémas cryptographiques basés sur la difficulté de ce problème. Cependant, les systèmes polynomiaux provenant d'applications telles que la cryptographie multivariée, ont souvent une structure additionnelle cachée. En particulier, nous donnons la première cryptanalyse connue du crypto-système « Extension Field Cancellation ». Nous travaillons sur le schéma à partir de deux aspects, d'abord nous montrons que les paramètres de challenge ne satisfont pas les 80 bits de sécurité revendiqués en utilisant les techniques de base Gröbner pour résoudre le système algébrique sous-jacent. Deuxièmement, en utilisant la structure des clés publiques, nous développons une nouvelle technique pour montrer que même en modifiant les paramètres du schéma, le schéma reste vulnérable aux attaques permettant de retrouver le secret. Nous montrons que la variante avec erreurs du problème de résolution d'un système d'équations est encore difficile à résoudre. Enfin, en utilisant ce nouveau problème pour concevoir un nouveau schéma multivarié d'échange de clés nous présentons un candidat qui a été soumis à la compétition Post-Quantique du NIST.
Mots clés : cryptographie, post-quantique, Multivariée, cryptage à clé publique, base de Gröbner, cryptanalyse algébrique, système polynomial avec erreurs, NIST.

Abstract

Polynomial system solving is one of the oldest and most important problems in computational mathematics and has many applications in computer science. It is intrinsically a hard problem with complexity at least single exponential in the number of variables. In this thesis, we focus on cryptographic schemes based on the hardness of this problem. In particular, we give the first known cryptanalysis of the Extension Field Cancellation cryptosystem. We work on the scheme from two aspects, first we show that the challenge parameters don't satisfy the 80 bits of security claimed by using Gröbner basis techniques to solve the underlying algebraic system. Secondly, using the structure of the public keys, we develop a new technique to show that even altering the parameters of the scheme still keeps the scheme vulnerable to attacks for recovering the hidden secret. We show that noisy variant of the problem of solving a system of equations is still hard to solve. Finally, using this new problem to design a new multivariate key-exchange scheme as a candidate for `NIST` Post Quantum Cryptographic Standards.
Keywords: Post-quantum, Cryptography, Multivariate, Public-key Encryption, Gröbner basis, Algebraic Cryptanalysis, Polynomial systems with Errors, NIST.