UNIVERSITY OF CINCINNATI

Date: 6-May-2010

I, John G Wagner ,

hereby submit this original work as part of the requirements for the degree of:

Doctor of Philosophy

in Mathematical Sciences

It is entitled:

Cryptanalysis of Rational Multivariate Public Key Cryptosystems

Student Signature: John G Wagner

This work and its defense approved by:

Committee Chair: Jintai Ding, PhD
Jintai Ding, PhD

Cryptanalysis of Rational Multivariate Public Key Cryptosystems

A dissertation submitted to the
Division of Research and Advanced Studies
of the University of Cincinnati

in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

in the Department of Mathematical Sciences
of the College of Arts and Sciences

2010

by

John Wagner

B.E.E., University of Dayton, 1987
M.E., University of Virginia, 1989

Committee Chair: Professor Jintai Ding

Abstract

In 1989, Tsujii, Fujioka, and Hirayama proposed a family of multivariate public key cryptosystems, where the public key is given as a set of multivariate rational functions of degree 4 [22]. We call these the Rational Multivariate Public Key Cryptosystems (RMPKC). These cryptosystems are constructed via composition of two quadratic rational maps into one quartic rational map, which becomes the public key. In this paper, we present a cryptanalysis of RMPKC.

This cryptanalysis demonstrates success against two separate problems in mathematics which are difficult to solve: factorization of maps and solving multivariate non-linear polynomial equations. We first perform a factorization of the public key quartic rational map into two components which are quadratic. We then attack each quadratic component, providing a way to solve the quadratic equations.

Our cryptanalysis is of the strong type. We take a public key and create a private key. The cryptanalyst can decrypt a message equally as fast as the owner of the original private key.

Our work involving the factorization of maps starts applying work published by Faugere and Perret, who set out to do basically the same thing. Their method, however, was insufficient to attack RMPKC. We enhance the method using projections to lower dimensions.

Our work involving the solution of quadratic equations is inspired by a thorough analysis of the structure of RMPKC and identification of weaknesses within.

To Lucas and Cynthia, for whom my love is infinite.

Acknowledgements

First of all, I want to acknowledge the importance of Dr. Jintai Ding's good work in the area of cryptography. Because of his leadership, the University of Cincinnati Math Department is full of opportunities for graduate students wanting to do research in this area. His successful matching of me to a perfectly suitable problem, his help in getting started on the problem, and his guidance on the written presentations of the research were each significant factors in my successful completion of this enormous task.

Secondly, I want to acknowledge the critical involvement of Dr. Dieter Schmidt and Dr. Timothy Hodges in their work with Jintai. Dieter's involvement has enabled cryptography in practice to be part of our department, part of my work, and part of the work of several other graduate students. Dieter, the cryptography group will not be the same without you. You will be missed. I also acknowledge Tim's contributions during his tenure as department chair and I am truly excited for the cryptography group to have Tim's involvement increase soon. The momentum sustained by all of the faculty involved enables cryptography to be a very attractive and productive area for graduate students both now and in the future.

Thirdly, I want to acknowledge the necessity for me to have the support that I did from so many fellow graduate students throughout the years. Working with Laura Bristol and Kate Pullin made passing the algebra prelim possible. Working more with Kate made passing the complex analysis prelim possible. Living with Dr. Jose Manuel Gomez inspired me to push harder and learn more. Working with Dr. Crystal Clough, Daniel Cabarcas, Dr. John Bayron Baena, and Zhijun Ying helped me learn finite fields and cryptography.

I also want to thank the rest of the faculty and staff of the mathematics department at UC. I have learned so much from my experience as a graduate student here, far more than what I learned in cryptography. I am a mathematician capable of contributing in countless areas of work, and I am a human with more experience, depth, and possibly more wisdom.

Lastly, I want to acknowledge all of the support and encouragement for completing this dissertation that I have received over past three years. The task was quite difficult while working full-time. While acknowledging that there are others who could be included, I'll end with the following list of people who made a difference: Lucas Wagner, Cynthia Wagner, Dr. Crystal Clough, Daniel Cabarcas, Dr. John Bayron Baena, Dr. Ruth Guterrez, Nancy Diemler, Monica Skrzelowski, Robert Hess, Susan Skene, Brian Smith, Terry Skaggs, Dr. James Osterburg, Dr. Mauricio Osorio, Ivonne Rivas, Theresa Wagner, Joseph Wagner, and Dr. Naomi Barnett.

Thanks to all of you.

Table of Contents

Abstract	iii
Dedication	v
Acknowledgements	vi
Chapter	
1. Introduction	1
1.1 Public Key Cryptography	1
1.2 RSA, Elliptic Curves, and Quantum Computers	4
1.3 Post-Quantum Public Key Cryptography	8
1.4 Multivariate Public Key Cryptography	12
1.4.1 Early Triangular Cryptosystems	13
1.4.2 Matsumoto-Imai	13
1.4.3 Further Work on Quadratic MPKCs	15
1.4.4 MPKC with Quartic Polynomials	17
1.4.5 Tractable Rational Maps	18
1.4.6 RMPKC	20
1.5 Our Research: Cryptanalysis of RMPKC	21
1.5.1 Motivation	21
1.5.2 Objectives	22
1.5.3 Challenges	23
1.5.4 Presentation	23
2. Notation	25
3. The RMPKC Cryptosystem	30
3.1 The public key	31
3.2 Encryption	33
3.3 The private key	33
3.4 Decryption	37
3.5 Division by Zero	38
3.6 Memory and Computational Requirements	40
3.7 Small-Scale Example	41
4. 2R	46
4.1 Cryptographic System 2R	46
4.2 Cryptanalysis of 2R	48

5. Cryptanalysis of RMPKC	54
5.1 Overview	54
5.2 Step 1: Homogenize the Public Key	65
5.3 Step 2: Find $\mathcal{F}$	71
5.3.1 Direct Application of the Partial Derivative Attack on $\hat{G} \circ \hat{F}$	72
5.3.2 Our Enhancement to the Partial Derivative Attack	77
5.4 Step 3: Find $F' \circ L'_1$	82
5.4.1 Find the Core Transformation of F'	84
5.4.2 Create an Inversion Process for the Core Transformation of F'	89
5.4.3 Find the Sequential Part of F'	91
5.5 Step 4: Find $\bar{G}'$	92
5.6 Step 5: Find $\bar{G}''$ and L'_2	94
5.7 Step 6: Find L'_3 and G'	96
5.8 Small-Scale Example	98
5.9 Experimental Results and Computational Complexity	110
6. Conclusions and Future Work	112
6.1 Summary of Accomplishments	112
6.2 Ideas for Future Research	112
6.2.1 Enhancements to RMPKC	112
6.2.2 Making the Cryptanalysis More Efficient	113
6.2.3 Expansion of the Cryptanalysis to Cover Odd Characteristic	114
6.2.4 Further Mathematical Analysis	114
Appendices	115
Bibliography	118

Chapter 1

Introduction

1.1 Public Key Cryptography

Cryptography is the study of methods to privately exchange information that gets passed from the sender to the intended receiver through means that risk exposure to third parties, from which the sender wants to maintain the information secret. Private key cryptography requires that the two parties wanting to exchange information secretly over non-secure channels must meet secretly beforehand to establish some sort of key, or encryption scheme (hence the name private key cryptography). The key enables them to encrypt and decrypt messages. We refer to the original message as plain text and we refer to the encrypted message as cipher text. If a third party intercepts the cipher text, the third party wouldn't understand it, unless they also could somehow decrypt it. To decrypt it the third party would either need access to the secret key, or some means to cryptanalyze, i.e. discover how to decrypt the cipher text without the private key.

Public key cryptography is cryptography that does not require the prior secret meeting between two parties to exchange a private key. Diffie and Hellman are credited for initiating the idea [2]. In public key cryptography, one party establishes a public key for encryption and a matching private key for decryption. The public key is made available publicly. The private key is held securely by its creator. Anyone wishing to send plain text to the owner

of the private key needs to access the matching public key and use it to encrypt the message into cipher text. The cipher text can be sent to the owner of the private key for decryption back into the original plain text. If a third party intercepts the cipher text, the third party cannot decrypt it (unless they gain access to the private key or successfully cryptanalyze the public key - private key pair).

The coming together of humans from across the globe via the world wide web highlights the need for public key cryptography. Two parties who meet on the internet might very well have the need to exchange information secretly. The most obvious example is party B providing some product or service for party A, and party A wanting to pay party B using a credit account. Party A would need to send the account number to party B, and would like to keep this account number secure from malicious third parties. Party B, the product or service provider, can create a public key - private key pair, publish the public key, thus enabling party A to send the account number securely, as described in the paragraph above. In this case, the account number is the plain text.

This internet example cited is only one example. The technology evolution we are experiencing today continues to create more communication needs, many of which can be enhanced by cryptography. For example, a public key - private key pair can be used to authenticate the originator of a document. In this case, let party A be the creator and holder of the private key. When party A wants to publish a document and maintain publicly that s/he is the originator, party A publishes the document, the public key, and a signature of the document. The signature is created by applying the private key to the document. Validation of the signature is performed by applying the public key to the signature. In this case, the signature is like the plain text and the original document is like the cipher text. Anyone with the publicly available public key can convert the signature into the original document, but only the owner of the private key can convert the document into a signature.

If a third party claims to be the originator of the original document - signature - public key triplet, party A can prove his/her authenticity by providing a signature for a second document which could be used to recreate the second document using the same public key previously published. The third party's claim would be exposed as false unless s/he too could produce a signature that would be converted into the second document via the previously established public key. In order to fool the world, the third party would have to be able to cryptanalyze the public key.

It is easy to see that cryptanalyzing a public key cryptosystem is inherently easier than cryptanalyzing a private key cryptosystem. The information available to the cryptanalyst is significantly different in the two cases. For the cryptanalysis in the private key case, the cryptanalyst has only intercepted cipher text. S/he would need additional information such as some example plain text - cipher text pairs, or possibly relevant information about the content of the plain text, together with matching cipher text. Because of the difficulty in cryptanalyzing private key cryptosystems, they are often very efficient.

The information available to a cryptanalyst of a public key cryptosystem is far greater. In this case, the cryptanalyst has full access to the public key! S/he can create as many plain text - cipher text pairs as s/he desires. For this reason public key cryptosystems are often used to communicate a private key cryptosystem. Party A creates a private key system and uses party B's public key to send to party B the private keys. The secret meeting between the two parties required for a private key cryptosystem is fulfilled by the exchange enabled by the public key system.

There exists a community of mathematicians and computer scientists dedicated to the exploration of cryptography. The work of this community relies on the development of secure cryptosystems that protect against cryptanalysis and development of powerful techniques of cryptanalysis. The primary objective of this community's work is to discover the secure

cryptosystems. However, if the community fails to lead the world in cryptanalysis, then it will not know when a cryptosystem is secure.

As described previously, cryptanalysis of a public key cryptosystem is performed with the public key available. Mathematically, we consider the public key a function that maps plain text to cipher text. The cryptanalyst attempts to invert that function. In strong cryptanalysis, the cryptanalyst actually creates an alternate private key, which could be used to convert any given cipher text back into the plain text that was used to create it. In weak cryptanalysis, the cryptanalyst develops a method that takes a cipher text and public key and works to discover the plain text that was used to create the cipher text. The primary difference here is that in the weak case, the cryptanalysis of a cipher text takes a much longer time. In the strong case, once an alternate private key is established, converting cipher text to plain text takes approximately just as long as it takes the holder of the original private key.

The research described by this dissertation is part of the cryptanalysis process embraced by the community. Our cryptanalysis is strong, resulting in a private key that is equivalent to the original private key. Appropriate follow up research could very well be to build on the ideas of the cryptosystem which gets attacked by our cryptanalysis and create a more secure cryptosystem impervious to our attacks and other known cryptanalytic methods.

1.2 RSA, Elliptic Curves, and Quantum Computers

In the late 1978, Rivest, Shamir, and Adelman published a public key cryptosystem, famously known as RSA [20]. RSA is based on the difficulty of converting large integers into their prime factors. RSA is relatively simple and quite elegant. The creation of an RSA cryptosystem begins with the selection of two positive prime integers, q_1 and q_2 . Let $Q = q_1q_2$. A third positive integer, e_1 , must be selected such that e_1 and $\phi(Q)$ are relatively

prime. Note that ϕ is the common Euler-Phi function, also known as the Totient function. $\phi(Q)$ is the count of the number of integers between 1 and Q which are relatively prime to Q , including 1 and excluding Q . Since Q is the product of two primes, $Q = q_1 q_2$, $\phi(Q) = (q_1 - 1)(q_2 - 1)$. Lastly, a fourth positive integer, e_2 must be identified such that $e_1 e_2 = j\phi(Q) + 1$ for some positive integer j .

The public key is identified by Q and e_1 . The private key is Q and e_2 . To encrypt plain text $X \in \{1, 2, \dots, Q - 1\}$ into cipher text $Y \in \{1, 2, \dots, Q - 1\}$ simply perform $Y = X^{e_1} \bmod Q$. To decrypt cipher text Y back into the plain text $Z \in \{1, 2, \dots, Q - 1\}$, simply perform $Z = Y^{e_2} \bmod Q$. It turns out that $Z = X$. This is true because integers in $\{1, 2, \dots, Q - 1\}$ fall into one of the following three categories:

1. multiples of q_1 : these integers form a group under multiplication of size $q_2 - 1$;
2. multiples of q_2 : these integers form a group under multiplication of size $q_1 - 1$;
3. integers relatively prime to Q : these integers form a group under multiplication of size $\phi(Q) = (q_1 - 1)(q_2 - 1)$.

In any of the three cases, the size of the group divides $\phi(Q)$. From basic group theory, if $X \in \mathcal{G}$, then $X^{|\mathcal{G}|} = 1$, so also $X^{j|\mathcal{G}|} = 1$ for any positive integer j (note $\mathcal{G}$ is the group in question and $|\mathcal{G}|$ is the size of the group). Therefore, regardless of which case X is in,

$$Y^{e_2} = X^{e_1 e_2} = X^{j\phi(Q)+1} = X^{j\phi(Q)} X = X.$$

The most obvious way to cryptanalyze RSA is to take the public key, Q and e_1 , and determine an e_2 to use for decryption. The difficulty in determining an e_2 is to ensure that $e_1 e_2 = j\phi(Q) + 1$ for some positive integer j . Of course for this you need $\phi(Q)$. The best way to get $\phi(Q)$ is to factor Q into $q_1 q_2$ because $\phi(Q) = (q_1 - 1)(q_2 - 1)$. Factoring Q is the difficult mathematics problem that makes RSA a viable cryptosystem.

The advancement of the capability to factor large primes over the years has produced two key consequences: 1) practical applications of RSA are made to use even larger primes, forcing cryptanalysis to take even longer, 2) additional requirements to the choice of primes, to thwart attacks using algorithms which rely on primes which have specific characteristics, and 3) the cryptography community has grown and expanded its research into more areas looking for other good options for cryptosystems.

Another early family of cryptosystems was based on the mathematical difficulty of computing the discrete logarithm. Consider the following where g, y , and x are all elements of some multiplicative group $\mathcal{G}$:

$$y = g^x.$$

Given g and x , it is easy to calculate y . But given y and g , it is much more difficult to calculate x . It is so difficult for many large groups that cryptosystems have been designed based on this problem. This mathematical difficulty is analogous to that of factoring primes, as described as critical to RSA. The structure of these cryptosystems are varied, but the discrete logarithm is at the heart of the cryptanalysis.

Elliptic curve cryptosystems are a family of discrete logarithm cryptosystems that have been applied to elliptic curves. Choose some finite field k of large characteristic and two elements from k , namely a and b . As long as the polynomial $x^3 + ax + b$ has neither a double nor a triple root, $y = x^3 + ax + b$ defines an elliptic curve. Points on the elliptic curve form a group $\mathcal{G}$ with an operation that is used for the cryptosystem. A generator of a subgroup of $\mathcal{G}$, preferably a generator of $\mathcal{G}$ itself, the order of this generator, and the order of $\mathcal{G}$, together with the original finite field and the elliptic curve, define the environment used for the cryptosystem. As in the family of cryptosystems based on the discrete log problem, the exact scheme used for encryption and decryption is quite varied [18].

When the elliptic curve is based on the real numbers, rather than a finite field, the group

operation has a very elegant geometric interpretation. The multiplication of two group members is found by drawing a line between those two points on the elliptic curve. The resulting product is the inverse of the third point intersected by the line. Using a finite field eliminates this geometric interpretation, but the actual calculation used in a practical implementation is just about the same for the finite field case as the real numbers case.

Advances in the algorithms to calculate the discrete logarithm, have forced implementations of these cryptosystems to get larger and avoid certain structures. Elliptic curve cryptosystems in some cases prove to be more efficient than RSA, but their future might very well be short.

In 1999, Shor published a paper, "Polynomial Time Algorithms for Prime Factorization and Discrete Logarithm on a Quantum Computer" that changed public key cryptography fundamentally, given the hope that a quantum computer can someday be constructed [21]. In his paper Shor explains algorithms for both prime factorization and the discrete logarithm calculation. With the current understanding of the quantum computer, which utilizes quantum states rather than binary states, all operations are reversible. If you can multiply, you can divide. If you can exponentiate, you can take a logarithm.

Furthermore, the two algorithms are very closely linked. The algorithm to factor a number n into its primes actually uses as its principal engine, an algorithm to compute the order of a random element x in the multiplicative group containing $\{ 1, 2, \dots, n-1 \}$. Call the order of the random element r and calculate $\gcd(x^{r/2} - 1, n)$. If r is even and if $x^{r/2} \not\equiv -1 \pmod{n}$, then the $\gcd(x^{r/2} - 1, n)$ is a non-trivial divisor of n . This is because $(x^{r/2} - 1)(x^{r/2} + 1) = x^r - 1 \equiv 0 \pmod{n}$. Given the random choice x and m equal to the number of distinct odd factors of n , the probability of finding a prime factor is greater than $1 - \frac{1}{2^{m-1}}$.

The time required to perform each of these difficult mathematical operations is so much

less using Shor's algorithms on quantum computers than what is possible today that cryptosystems relying on the difficulty of either prime factorization or discrete logarithm will be rendered insecure when quantum computers are available. Of course we have some time before quantum computers become available. While the developers of quantum computers progress, so must the cryptography community progress to develop cryptosystems that will be robust even in the post-quantum computer age.

1.3 Post-Quantum Public Key Cryptography

Given the position that quantum computers running Shor's algorithms will render insecure cryptosystems based upon prime factorization and the discrete logarithm, other families of cryptosystems are gaining more attention. One of those families is hashed-based cryptosystems.

Hash-based cryptosystem, as the name implies, are based on hash functions. A hash function is a general term for well-defined functions that typically map several elements of the domain space to the range space. Hash functions are extremely difficult to invert, meaning to find a pre-image in the domain space of a given value from the range space. Hash functions have an extensive list of applications in computer programming. Many of the more straightforward applications are related to data storage and retrieval. More sophisticated applications can be used for data analysis, for example to find two similar pieces of data.

Hash functions can be used in cryptography for signature schemes, like the Lamport Signature [9]. In this signature scheme, the document to be signed is coded in a way to identify indices in the private key. Let $(m_1, m_2, \dots, m_n)$ be the coded document, where each m_i points to an index of an array, specifically the column of the array where i identifies the row (typically the column options are $\{0, 1\}$). The array associated with the private key is made up of randomly chosen elements from the domain space of the hash function. An

array of the same size associated with the public key is made up of the images under the chosen hash function of the corresponding array elements of the private key. To sign the coded document, simply identify the private key array elements per the encoded document and provide them publicly with the document and the hash function. Authentication of the signature is simple: perform the hash function on the signature and compare the result to the public key array elements identified by the encoded document.

Unfortunately, this signature scheme as presented is disposable; each signed document provides n pairs of values from the private key. With access to several document - signature pairs a cryptanalyst could reconstruct the private key directly. Other schemes, such as those employing Merkle trees, build on the Lamport Signature and are capable of repeated uses [12].

Another family of cryptosystems potentially viable in the post-quantum world are based on error-correcting codes. For our purposes, consider the generators of the error correcting codes as functions that map plain text to cipher text, as in an encryption scheme. The size of the cipher text is much larger than the size of the plain text, specifically because it contains redundant information. The function is designed so that a pre-image of the cipher text can be found, even if there are errors in the cipher text. Usually these systems operate on a stream of bits (binary), so an error is the switching between a 0 and a 1.

Consider a public key defined as the composition of the following three functions in order: one linear transformation, one error-correcting code generator, and one permutation. The linear transformation and the permutation effectively hide the structure of the error-correcting code generator. The resultant composition of functions is linear, but not easily invertible. Along with the public key composition of functions, the maximum number of errors t acceptable to the error-correcting code must be supplied. The private key includes the inverse of both the linear transformation and the permutation, and the ability to find

the pre-image of the error-correcting code generator.

To encrypt a message of bits, the sender evaluates the public key function, where the message provides the values for the variables, and randomly adds up to t bit errors to the result. This result with errors is the cipher text.

To decrypt the cipher text, the holder of the private key begins by applying the inverse of the permutation to the cipher text. The errors created in decryption get permuted and continue to affect the exact same number of bits, which is under the limit of the capabilities of the error-correcting code. Therefore, the next step of decryption, finding the pre-image of the error-correcting code generator can be successful. The last step, of course, is to apply the inverse of the linear transformation which provides the original plain text.

Like all other cryptosystems, these based on error-correcting codes have suffered attempts of cryptanalysis. Enhancements to Robert McEliece's original scheme published in 1978 have been deemed to be less efficient than the original. The original parameters regarding the size of the cryptosystem have been declared insufficient to avoid the latest attempt of cryptanalysis by Bernstein, Lange and Peters [11] [1].

Lattices are an interesting mathematical construct that have led rise to yet another family of cryptosystems with post-quantum potential. A lattice is a subset of $\mathcal{R}^n$ defined by a basis for $\mathcal{R}^n$. Various bases may define the same lattice. Given a basis for $\mathcal{R}^n, \{ b_1, b_2, \dots, b_n \}$, a lattice $\mathcal{L}$ is the following set

$$\mathcal{L} = \{ z b_i : 1 \leq i \leq n, z \in \mathcal{Z} \}$$

which forms an additive subgroup of $\mathcal{R}^n$.

The two mathematically difficult problems with lattices are called the shortest vector problem and the closest vector problem. The difficulty of these problems is dependent upon the basis used to define the lattice. In encryption schemes, the private key consists of a 'good' basis, that is one with nearly orthogonal and small vectors. The definition of the

lattice used in the public key is a not-so-good basis, created by transforming the good basis in a way to preserve the lattice [19].

The plain text space is a square subset of $\mathcal{Z}^n$, like $\mathcal{W}^n$, where $\mathcal{W} = \mathcal{Z} \cap [-w, w]$ for some positive integer w . Let's consider the famous GGH encryption scheme which was developed in 1997 [5].

Let B be an invertible matrix defined by a good basis for a lattice $\mathcal{L}$. Let U be a unimodal matrix (also invertible). The private key is defined by B and U . The public key is the composition $B \circ U$, i.e. BU . Let m be the plain text; the cipher text is $BU(m) + e$ where e is some small error vector in $\mathcal{Z}^n$. Note that we expect the cipher text to not be in the lattice.

To decrypt, the holder of the private key applies B^{-1} to the cipher text, yielding

$$B^{-1}(BU(m) + e) = U(m) + B^{-1}(e).$$

At this point, an algorithm must be employed to find the closest vector to $U(m) + B^{-1}(e)$, hopefully $U(m)$. At which point U^{-1} is applied to yield m .

The algorithm mentioned to find the closest vector will work because of having the good basis related to B . The cryptanalyst has access to BU , which is not so good and thus not so helpful for this task.

This basic lattice-based cryptosystem is not secure but has led to more variations and this family of cryptosystems continues to be considered a viable option in the post-quantum world [14].

The last family of cryptosystems for us to consider is the polynomial-based family, known as multivariate public key cryptography (MPKC). MPKC is possibly the most exciting family, and the one in which our research lies. The following section will discuss this family and show how the work of various research efforts have led to the specific cryptosystem of which our cryptanalysis attacks.

1.4 Multivariate Public Key Cryptography

Multivariate public key cryptography (RPKC) uses as the public key multivariate polynomials. Inverting multivariate polynomials of degree greater than one is mathematically very difficult. In most cases, the polynomials used are quadratic. There are other variants, as we will see, but the use of higher degree polynomials is typically avoided because of the resultant dramatic increase to the size of the public key. The larger the size of the public key, the greater is the time required to download the public key. Additionally, the computational time required to use the public key to convert plain text to cipher text is also dramatically increased as the degree of the polynomial increases.

The most common approach used in developing a multivariate public key cryptosystem is to define a set of polynomials in a way that the inversion function (i.e. the private key) is determined in the process of defining the set of polynomials. Often that approach leads to the composition of three maps in the formation of the public key. The central non-linear map is the tricky one, which presents the mathematical difficulty of inversion, and which is composed of several polynomials of degree two (or higher). The other two maps are simple invertible linear maps that serve the important function of camouflaging the structure of the non-linear map.

Consider the following public key composed of the three maps L_1, F, L_2 :

$$\begin{array}{ccccccc} & L_1 & & F & & L_2 & \\ k^n & \longrightarrow & k^n & \longrightarrow & k^n & \longrightarrow & k^n \end{array}$$

where k is a finite field, k^n is an n -dimensional vector space over k , $X \in k^n$ is the plain text, $Y \in k^n$ is the cipher text, and $L_2 \circ F \circ L_1$ is the public key. $Y = L_2 \circ F \circ L_1(X)$. F can be based on relatively simply structured quadratic polynomials. But the variables of those simply structured polynomials get completely mixed up by the map L_1 . Then L_2 mixes up the resulting polynomials, $F \circ L_1$. In this way the cryptanalyst might know about

the structure of F , but given the public key, which is the completely composed function, $L_2 \circ F \circ L_1$, the cryptanalyst has a difficult time figuring out exactly what F is.

1.4.1 Early Triangular Cryptosystems

In the 1980's cryptosystems were developed that employed a triangular structure, as related to the construction of the component polynomials of F . For a cryptosystem that maps k^n to k^n , consider F to be composed of one polynomial, f_1 , linear in x_1 only, and $n - 1$ quadratic polynomials, $f_2, f_3, \dots, f_n$. For each $2 \leq i \leq n$, f_i is linear in x_i , quadratic in $x_1, x_2, \dots, x_{i-1}$, and independent of $x_{i+1}, x_{i+2}, \dots, x_n$.

Inversion of such an F is simple and very fast. You first solve f_1 for x_1 . Then knowing x_1 , you solve f_2 , which is linear in x_2 , for x_2 . You then iterate $i = 3, 4, \dots, n$ to solve, one at a time, for $x_3, x_4, \dots, x_n$. At each iterative step, the polynomial you solve is linear in the variable for which you are solving.

Later, in 1999, Moh published a paper describing the Tame Transformation Method (TTM), which builds on the early triangular structure by composing two such functions, one decrypted sequentially from the first component to the last, and the other one decrypted sequentially from the last component to the first [13]. Courtois and Goubin cryptanalyzed TTM employing an approach they call MinRank [6].

1.4.2 Matsumoto-Imai

In 1988, Matsumoto and Imai published their cryptosystem, which we call MI [10]. Their approach was simple and extremely elegant. The central function F is originally defined not on the vector space k^n but rather on the extension field of degree n over k (characteristic 2), which we'll call K . Note that a natural isomorphism exists between k and K that makes transformation of an element from one to the other nearly transparent. Also, univariate polynomials over K can be expressed as multivariate polynomials over k^n .

Exponentiation in K has an important property as expressed in k^n when the exponent is a multiple of the size of the smaller field k . Consider $A \in K$ expressed as $a_0 + a_1w + \dots + a_{n-1}w^{n-1}$ where each $a_i \in k$, w is the dummy variable, as in the polynomial ring $k[w]$, and K is a degree n extension over k and the order of k is q . Now consider $A^q = (a_0 + a_1w + \dots + a_{n-1}w^{n-1})^q$. Expansion of a polynomial raised to the q power includes terms involving each original monomial raised to the q power as well as $n^q - n$ cross terms each scaled by a multiple of q . These cross terms disappear because q is a multiple of the characteristic of the fields, and thus as a scalar in multiplication, $q = 0$. Thus

$$A^q = (a_0 + a_1w + \dots + a_{n-1}w^{n-1})^q = a_0^q + a_1^q w^q + \dots + a_{n-1}^q w^{(n-1)q}.$$

Furthermore, $a_i^q = a_i$ for each relevant i so $A^q = a_0 + a_1w^q + \dots + a_{n-1}w^{(n-1)q}$. Each power of w greater than n gets transformed into an element of K expressed as a polynomial of degree less than n with constant coefficients. The resulting expression for A^q in k^n is linear!

MI uses as the central map $F : X \longrightarrow X^{q^i+1} = X^{q^i}X$ which is a multiplication of X to another element expressed as a linear transformation of X . The result is a quadratic polynomial when expressed as a function over k^n .

The public key in MI is the composition of this F with two linear transformations as in $L_2 \circ F \circ L_1$. Encryption is the operation of the public key on the plain text, which is an element of k^n .

The private key is the inverse of each of the two linear transformations and t . Decryption is performed by applying the inverse of L_2 to the cipher text, then raising the results to the power t (this operation being performed over the larger field K). The result of this operation gets transformed by the inverse of L_1 into the original plain text.

As mentioned above, $F^{-1} : X \longrightarrow X^t$ is the decryption subprocess related to the central map F . In order for this to result in the correct inversion, we need $t(q^i + 1) = 1 \text{ mod}(q^n - 1)$.

So the choice of i needs to be such to have $\gcd(q^i + 1, q^n - 1) = 1$.

1.4.3 Further Work on Quadratic MPKCs

The revolutionary cryptosystem MI lasted for seven years, until in 1995, Patarin successfully performed cryptanalysis on it [15]. Patarin discovered equations based on the map F which, when given the cipher text Y , become linear in the variables related to the plain text X . He found that enough of these equations were linearly dependent to fully decrypt Y into X without access to the private key. These equations are now popularly known as linearization equations.

Confidence in the viability of MPKCs and the elegance of MI led to several enhancements and of course, subsequent attacks. We discuss some of these enhancements here for they may spark ideas for enhancements to the cryptosystem of our research, RMPKC.

MI+ is the name given to an enhancement to MI developed by adding additional randomly generated polynomials to the central non-linear map F . These additional polynomials further cloud the structure of F in the public key.

An analogo to MI+ is MI-, which works as its name suggests. MI- actually removes some of the quadratics of the central map. The resulting public key is useful for signature schemes but not for encryption - decryption, since the pre-image of F is non-unique [4].

A generalization of the big field equation used to form F is the key to another cryptosystem named the Hidden Field Equation Cryptosystem (HFE) [16]. The function F becomes the following:

$$f : X \longrightarrow \sum_{i=1}^{n_i} \sum_{j=1}^{n_j} a_{i,j} X^{q^i + q^j} + \sum_{i=1}^{m_i} b_i X^{q^i} + c.$$

The resulting public key still contains only quadratic polynomials over k^n . The drawback to HFE is in decryption. The super simple decryption of MI is lost. For HFE, the decryption process requires a sophisticated algorithm. Typically the Berlekamp algorithm is used. The

size of the largest power in the big field expression for F is the key factor in determining the expected computational complexity of decryption, in number of operations.

Yet another cryptosystem born out of MI is called Oil and Vinegar [17]. Like MI-, it is a signature scheme only, and relies on the domain space of F being larger than the range space. The domain space is divided into oil variables and vinegar variables. F is linear in the oil variables and quadratic in the vinegar variables. A pre-image of F is found by randomly choosing vinegar variables and solving for the oil variables. Oil and Vinegar was partially inspired by the cryptanalysis involving linearization equations.

One more enhancement method in this family of cryptosystem is called perturbation [3]. The resultant cryptosystem is named PMI (perturbed Matsumoto-Imai). This specific method of perturbation interacts within F . The quadratic polynomials of F expressed over k^n get additional terms added into the polynomial. Decryption includes guessing at the values of the variables related to the additional terms added. For each guess, the additional terms are computed and subtracted from the vector that is being decrypted. The resultant MI system is solved and the solution is tested against the original guess. If it fails, another guess must be made.

These enhancements to MI can also be used together. For example HFEV is a cryptosystem created by adding vinegar variables to an HFE system. PMI+ is another cryptosystem created by adding random quadratics as additional polynomials to the PMI scheme. Many more efforts, both in developing cryptosystems and in cryptanalyzing them, have been made and published. A more exhaustive review of them would be inappropriate here. A more detailed treatment will be made of the developments of cryptography that led more directly to the research explained in this dissertation [4].

1.4.4 MPKC with Quartic Polynomials

In 1997, Patarin and Goubin designed cryptosystems based on two rounds of non-linear functions like F [7] [8]. Because of the two rounds, these cryptosystems are known as '2R'. We'll call the non-linear maps F and G . They use three linear maps to hide the structures of F and G . The public key is the composition $L_3 \circ G \circ L_2 \circ F \circ L_1$. The polynomials which express each of F and G are quadratic, so the polynomials of the public key are quartic.

Using quartic polynomials requires more memory to store the keys, more time to transfer the public key, and more time to compute the cipher text using the public key. However, the idea to compose two non-linear functions to increase the difficulty of discovering each is very interesting.

This approach actually introduces another hard mathematics problem into MPKCs: factoring maps. The first hard problem was solving non-linear multivariate polynomials. The composition of the F and G into one map with quartic polynomials presents the cryptanalyst with the possibility of using the factorization of maps to decompose the map based on quartic polynomials into maps based on quadratic polynomials.

This distinction of 2R from MPKCs based on quadratic polynomials is fundamental. In the other cryptosystems we saw a non-linear map composed with two linear transformations. But these linear transformations are not the same as composing the non-linear map with another non-linear map. In $L_2 \circ F \circ L_1$, L_1 really only serves to change the representation of the domain space by changing the basis. L_2 merely mixes the polynomials of F .

This factorization of maps as in 2R is analogous to the factorization of integers into prime factors. Recall in RSA that two large primes q_1 and q_2 are multiplied together to form Q , which is part of the public key. In cryptanalysis, factoring Q into q_1 and q_2 effectively cracks the code. The impetus behind 2R is really about using the factorization problem once again for a cryptosystem because of so many advances in the cryptanalyses of systems based on

quadratic polynomials.

In 1999, Ye, Lam, and Dai published a cryptanalysis of 2R [24]. They succeed in factoring the composite map of 2R into its factors. Consider the composite map $G \circ F$. In the cryptosystem attacked, each of G and F is composed of n quadratic polynomials. The result, $G \circ F$, is composed of n quartic polynomials. We believe that this group's work is ground-breaking in this area of the factorization of maps.

The method employed by this group is called the partial derivative attack. The basic premise for the attack is

$$Span \left\{ \frac{\partial}{\partial x_i} g_j \circ F(x_1, \dots, x_n) : 1 \leq i, j \leq n \right\} \subseteq Span \{ x_i f_j(x_1, \dots, x_n) : 1 \leq i, j \leq n \} .$$

This result is used to find $Span \{ f_j(x_1, x_2, \dots, x_n) : 1 \leq j \leq n \}$.

This method uses each of the elements of the Jacobian Matrix, J , of $G \circ F$, the partial derivatives. In the Jacobian Conjecture, still an open question for $n > 2$, the determinant of J is related to the invertibility of F .

We aim to improve on Ye, Lam, and Dai's work of factoring of maps. As you will see, our research relies heavily on their work. Their method is important enough for us to dedicate an entire chapter to explain both the cryptosystem and their cryptanalysis.

1.4.5 Tractable Rational Maps

In 2005, Wang, Hu, Lai, Chou, and Yang published a paper describing a cryptosystem based upon what they call tractable rational maps (TRM) [23]. They employ a triangular method utilizing rational maps, as opposed to polynomial maps. The construction of TRM is close enough to the construction of RMPKC to warrant a more formal explanation than what we've given for other examples of cryptosystems. So let k be a finite field and consider

$F : k^n \longrightarrow k^n$ be a tractable rational map.

$$F(x_1, x_2, \dots, x_n) = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix}$$

where

$$f_1(x_1, x_2, \dots, x_n) = r_1(x_1)$$

and for $2 \leq i \leq n$,

$$f_i(x_1, x_2, \dots, x_n) = r_i(x_i) \left[\frac{p_{i,1}(x_1, x_2, \dots, x_{i-1})}{q_{i,1}(x_1, x_2, \dots, x_{i-1})} \right] + \left[\frac{p_{i,2}(x_1, x_2, \dots, x_{i-1})}{q_{i,2}(x_1, x_2, \dots, x_{i-1})} \right].$$

For each $1 \leq i \leq n$, $r_i(x_i)$ is an invertible function, $r_i : k \longrightarrow k$, and $p_{i,1}, p_{i,2}, q_{i,1}$, and $q_{i,2}$ are each polynomials. Typically such a map F will be composed with linear transformations as in our other examples.

Decryption of the non-linear function F in the TRM cryptosystem is performed almost exactly as explained in the section above entitled ‘Early Triangular Cryptosystems’. Iteratively, we solve for x_1 using f_1 , then we solve for x_2 using f_2 , and so on, until we lastly solve for x_n using f_n .

The authors of TRM add complexity to this base case for F . Firstly, they allow for the use of k – *algebras* in the structure of k^n . For example, we can consider $(x_1, x_2, x_3, x_4) \in k^4$ as $X_1 \in GF(k^4)$, possibly with $X_1 = x_1 + x_2z + x_3z^2 + x_4z^3$ in the realization of $GF(k^4)$ as $k[z] \text{ mod } \psi$ where ψ is a degree four irreducible polynomial over k . This mixing up of base algebraic systems is a nice generalization to their base case.

The second added complexity comes from composition of functions like F in such a way that the resulting public key can be expressed using quadratic polynomials (actually rational expressions). This new resultant class of cryptosystems appears to show some promise.

Furthermore, an advantage of using rational maps in the multivariate public key cryptosystem is that you can represent efficiently maps from k^n to k^n that would be of much higher degree if you were to represent them with polynomial expressions. Keeping the expressions for the polynomials of the numerators and denominators to quadratic causes the resultant public key to be quite small.

1.4.6 RMPKC

In 1989, even before Patarin and Goubin (1997) published 2R and Wang, Hu, Lai, Chou, and Yang (2005) published TRM, Tsujii, Fujioka, and Hirayama published a paper entitled "Generalization of the Public-Key Cryptosystem Based on the Difficulty of Solving a System of Non-linear Equations" [22]. The cryptosystem described, which we call the Rational Multivariate Public Key Cryptosystem (RMPKC), employs aspects of both 2R and TRM. RMPKC employs rational expressions for the component functions as TRM does, and RMPKC employs a composition of two rounds of non-linear functions, resulting in expressions for functions involving quartic polynomials, as 2R does.

However, RMPKC hasn't drawn the same attention as 2R and TRM did. The publishing of our cryptanalysis of RMPKC is the first.

RMPKC will be presented in detail in chapter 3. Here we will mention some historical highlights that mark the creation of RMPKC. Firstly, each of the two primary non-linear maps of RMPKC is quite similar to the base case of F in TRM. However there are some key differences:

1. The denominator polynomials of RMPKC are one and the same, i.e. $q_{2,1} = q_{2,2} = q_{3,1} = q_{3,2} = \dots = q_{n,1} = q_{n,2}$,
2. The initial step in decryption involves two component rational expressions, where in TRM only one rational expression is employed,

3. In RMPKC, $r_i(x_i) = x_i$ for each $1 \leq i \leq n$.

Secondly, RMPKC employs an efficiency improvement by defining the public key as a map from k^n to k^n , but then expanding the map to function from K^n to K^n where K is a finite extension field of k . This expansion is 'free', because $k \subseteq K$. This efficiency improvement reduces the size of the public key, since the coefficients of the polynomial expressions are elements of k rather than K , but it does not reduce the computational time required to encrypt a message.

Lastly, RMPKC chooses values for the parameters n and for the size of k for increased efficiency. The suggested implementation uses $n = 5$ and $|k| = 2^8$ ($|K| = 2^{32}$). These choices decrease significantly the size of the public key and the computational time required for encryption.

1.5 Our Research: Cryptanalysis of RMPKC

Our chosen research described here is a cryptanalysis of RMPKC, as described by Tsujii, Fujioka, and Hirayama in 1989.

1.5.1 Motivation

The complete cryptanalysis of RMPKC will employ tactics to address both hard mathematical problems related to the MPKCs involving quartic polynomials: solving non-linear equations and factoring maps. Factoring maps will happen first, to break down the composite map into two maps that form it. Solving the non-linear equations happens next, by finding a way to invert the resultant maps just mentioned. Our contribution is fundamentally different than the attack on 2R for two reasons: 1) RMPKC uses rational functions, not polynomials, and 2) even when analyzing the polynomial expressions as we do, the partial derivative attack fails and requires our enhancement.

Our cryptanalysis on RMPKC puts to bed any question about the viability of RMPKC.

As it was designed, it is clearly insecure. Furthermore, the exercise of performing the cryptanalysis provides insight into further work in both developing better cryptosystems and performing cryptanalysis.

1.5.2 Objectives

The primary objective of the research was to implement, in Magma, a cryptanalysis of RMPKC as described by Tsujii, Fujioka, and Hirayama in 1989. Initially, it was thought that the partial derivative attack, as described by Ye, Lam, and Dai in 1999, would provide most of the firepower necessary for the cryptanalysis. As we will see, this is not the case. To fulfill the primary objective required a significant enhancement to the partial derivative attack. We are successful in meeting this objective.

A secondary objective was to make the cryptanalysis strong, that is, taking an RMPKC public key and developing an alternate private key to use for decryption. We do succeed in creating such an alternate private key. Our alternate private key is structured and used exactly like the original private key used in the creation of the public key.

An additional secondary objective was to expand our cryptanalysis to attack RMPKCs of greater size than that proposed by the authors in 1989. Specifically, the proposed RMPKC for implementation in 1989 included a small field k of size 2^8 and a 5-dimensional vector space over that field for the private and public keys. The larger field K used in the plain text and cipher text spaces was of size 2^{32} . Our expansion of the RMPKCs to attack maintained the same field size (and characteristic) and increased n up to 14. We stopped at $n = 14$ because the run time required on the computer available was becoming inconvenient. In completing this attack on larger systems, we found that our enhancement to the partial derivative attack needed to be generalized.

1.5.3 Challenges

The greatest challenges overcome in this research can be grouped into two areas: 1) how to enhance the partial derivative attack to get it to work on RMPKC, and 2) how to work through all of the obstacles met in the effort to determine non-linear functions structured as those used to form the RMPKC public key.

The enhancement to the partial derivative attack uses a projection of vector spaces onto vector spaces of lower dimension. The attack is completed on several cases of the projection, and the result is built back up into the higher dimension vector space.

In the effort to reconstruct non-linear maps as part of the alternate private key in such a way so that the private key would behave exactly like the private key held by its legitimate owner presented several obstacles. Finding a valid core transformation and finding the exact denominator for the map F were the two most difficult of these obstacles. The solution to both of these obstacles involved the weakness of RMPKC related to the linear subspaces in the span of the polynomials used in the construction of the two non-linear maps, F and G .

We found this research exciting and interesting. We realize that reading this dissertation will by no means be as thrilling as it was to successfully crack a published cryptosystem, but we hope that it might inspire interest and knowledge for further work.

1.5.4 Presentation

This presentation is organized into chapters as follows:

1. Introduction
2. Notation

We explain the notational conventions used throughout the presentation.

3. The RMPKC Cryptosystem

We spell out the precise definition of the cryptosystem that our cryptanalysis attacks.

4. 2R

We explain both the cryptosystem 2R and its cryptanalysis.

5. Cryptanalysis of RMPKC

We explain in detail our cryptanalysis of RMPKC.

6. Conclusions and Next Steps

We summarize our work and lay out suggested follow up research.

7. Appendices

We provide the Magma code used in this cryptanalysis as well as instructions on how to run it. Unfortunately, executing this code requires an installation of Magma.

Chapter 2

Notation

One of the challenges of presenting the mathematics of this research is to use notation that helps, rather than confuses, the reader. In this chapter we try to lay out the basics of the notation employed.

The letters k and K are only used to represent the finite fields used in RMPKC.

The letters i, j, r, m , and n are always natural numbers. i, j, r, s , and t always represent counting variables and subscripted indicies; m and n always represent fixed quantities. n is always used as the parameter of the RMPKC that defines number of dimensions employed.

Much of the mathematics included is about what we are calling multivariate polynomial maps. For the most part, the letters P, F, G , and L will be used for these. Here is an example:

$$F : k^n \longrightarrow k^{n+1}, F = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{n+1} \end{pmatrix}$$

where $\forall 1 \leq i \leq n+1, f_i : k^n \longrightarrow k$ where

$$f_i(x_1, x_2, \dots, x_n) = \sum_{j=1}^n \sum_{k=j}^n f_{i[j,k]} x_j x_k + \sum_{k=1}^n f_{i[n+1,k]} x_k + f_{i[n+1,n+1]}$$

where for all relevant values of i, j , and k , $f_{i[j,k]}$ is a coefficient from k .

This example of F uses multivariate quadratic polynomials to define each component map, f_i . We shall therefore refer to F as a multivariate quadratic polynomial map; sometimes this may be shortened to quadratic map, since almost all of the maps we consider in this research are multivariate polynomial maps. The choice of subscripts for the linear and constant terms are chosen specifically to enable the compaction of the notation required for the homogenization of F . We shall see this shortly.

At times we will list only a subset of the parameters of a polynomial map. We do this intentionally. Even though the map is defined on k^n , or k^{n+1} as its domain, the map is independent of the parameters left out. The purpose of our choice to omit the irrelevant parameters is to demonstrate that they are irrelevant.

G will also refer to a multivariate quadratic polynomial map defined as F was defined, using g to denote the component maps of G , just like f was used to denote the component maps of F .

P is used to denote a similarly constructed multivariate quartic polynomial map. Note that the subscripts on the coefficients in the quartic polynomials associated with P will require subscripts like $p_{i[j,k,r,s]}$.

L_1, L_2 , and L_3 are used to refer to multivariate linear polynomial maps. Their construction will be analogous to what we've shown for F above. At times we may refer to these multivariate linear polynomial maps as linear transformations.

As the mathematics require us to alter these maps in various ways, we now establish a convention to provide some consistency in the notation for the alterations.

First let's consider the $\sim$ symbol, as in $\tilde{F}$. The $\sim$ symbol will always refer to a map that has been altered such that the domain space will increase from k^n to k^{n+1} . For the quadratic and quartic cases, namely F, G , and P , we will have

$$F, G, P : k^n \longrightarrow k^{n+1}, \quad \tilde{F}, \tilde{G}, \tilde{P} : k^{n+1} \longrightarrow k^{n+1}.$$

$\tilde{F}$, $\tilde{G}$, and $\tilde{P}$ will represent the homogenized forms of F , G , and P , respectively. Here is

an example: $\tilde{F} = \begin{pmatrix} \tilde{f}_1 \\ \tilde{f}_2 \\ \vdots \\ \tilde{f}_{n+1} \end{pmatrix}$ where

$$\begin{aligned} \forall 1 \leq i \leq n+1, \quad \tilde{f}_i(x_1, x_2, \dots, x_{n+1}) &= x_{n+1}^2 f_i\left(\frac{x_1}{x_{n+1}}, \frac{x_2}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}}\right) \\ &= \sum_{j=1}^n \sum_{k=j}^n f_{i[j,k]} x_{n+1}^2 \frac{x_j}{x_{n+1}} \frac{x_k}{x_{n+1}} + \sum_{k=1}^n f_{i[n+1,k]} x_{n+1}^2 \frac{x_k}{x_{n+1}} + f_{i[n+1,n+1]} x_{n+1}^2 \\ &= \sum_{j=1}^n \sum_{k=j}^n f_{i[j,k]} x_j x_k + \sum_{k=1}^n f_{i[n+1,k]} x_{n+1} x_k + f_{i[n+1,n+1]} x_{n+1}^2 \\ &= \sum_{j=1}^{n+1} \sum_{k=j}^{n+1} f_{i[j,k]} x_j x_k. \end{aligned}$$

This compacted form (under one summation) is possible because of the specific choices for the indices on the linear and constant terms, as we mentioned previously.

For the linear cases, namely L_1 , L_2 , and L_3 , we will have

$$L_1, L_2, L_3 : k^n \longrightarrow k^n, \quad \tilde{L}_1, \tilde{L}_2, \tilde{L}_3 : k^{n+1} \longrightarrow k^{n+1}.$$

The alteration for each of them will always be the following, for $s = 1, 2, 3$:

$$L_s = \begin{pmatrix} L_{s,1} \\ L_{s,2} \\ \vdots \\ L_{s,n} \end{pmatrix} \text{ where } \forall 1 \leq i \leq n, \quad L_{s,i}(x_1, x_2, \dots, x_n) = \sum_{j=1}^n l_{s,i[j]} x_j \text{ where for all relevant}$$

values of s, i , and j , $l_{s,i[j]}$ is a coefficient from k . Note that L is homogeneous. Now $\tilde{L}_s$ will be defined as follows:

$$\tilde{L}_s = \begin{pmatrix} \tilde{L}_{s,1} \\ \tilde{L}_{s,2} \\ \vdots \\ \tilde{L}_{s,n+1} \end{pmatrix}$$

where

$$\forall 1 \leq i \leq n, \tilde{L}_{s,i}(x_1, x_2, \dots, x_{n+1}) = \sum_{j=1}^n l_{s,i[j]} x_j$$

where for all relevant values of s, i , and j , $l_{s,i[j]}$ is the same coefficient shown above for L_s ; finally

$$\tilde{L}_{s,n+1}(x_1, x_2, \dots, x_{n+1}) = x_{n+1}.$$

Our next special symbol to consider will be the $\bar{}$ symbol, as in $\bar{F}$ and $\bar{G}$. It will only be used for these quadratic maps. $\bar{F}$ will denote F composed with some linear transformation. We will use $\bar{F} = F \circ L_1$ where $\bar{F} : k^n \longrightarrow k^{n+1}$.

The $\hat{}$ symbol, as in $\hat{G}$, will be used to denote the composition of a homogenized version of the quadratic map with a linear transformation. An example is $\hat{G} = \tilde{G} \circ \tilde{L}_2$, where $\hat{G} : k^{n+1} \longrightarrow k^{n+1}$.

Another map used we'll call the division map, for obvious reasons. It will take the name ϕ .

$$\phi : k^{n+1} \longrightarrow k^n, \phi(x_1, x_2, \dots, x_{n+1}) = \begin{pmatrix} \frac{x_1}{x_{n+1}} \\ \frac{x_2}{x_{n+1}} \\ \vdots \\ \frac{x_n}{x_{n+1}} \end{pmatrix}, x_{n+1} \neq 0.$$

The prime symbol, $'$, as in F' , will denote maps defined specifically for the cryptanalysis. Without the prime symbol, a map refers to the composition of the public key. In the cryptanalysis, we have access to the exact public key, but not of the maps that together form the composition of the public key. We do have knowledge of the structure of these maps, and we will use an understanding of these structures to perform the cryptanalysis.

Consider the public key as defined by

$$\phi \circ P = L_3 \circ \phi \circ G \circ L_2 \circ \phi \circ F \circ L_1.$$

We know that each of L_1, L_2, L_3, G , and F exist and we know the general structure of

each, but we don't know (and we'll never know), the exact formulation of any of them. Of course we know ϕ exactly. However, in the cryptanalysis, we create exact formulations for each of L'_1, L'_2, L'_3, F' , and G' such that

$$\phi \circ P = L'_3 \circ \phi \circ G' \circ L'_2 \circ \phi \circ F' \circ L'_1.$$

So we will consistently use the prime symbol to denote maps for which we create exact formulas. The prime symbol is also used in conjunction with the $\tilde{\cdot}$, $\bar{\cdot}$, and $\wedge$ symbols in the most obvious way (e.g. $\bar{G}' = \tilde{L}'_3 \circ G' \circ L'_2$).

We sincerely hope that this explanation of the notation used eases the work required to sift through the mathematics, in which the notation can become cumbersome.

Chapter 3

The RMPKC Cryptosystem

In this section, we will present the design of the rational multivariate public key cryptosystem [22].

Let k be a finite field of characteristic 2 and K be an extension field over k . Let, k^n be the n -dimensional vector space over k and K^n be the n -dimensional vector space over K .

The public key converts plain text to cipher text. In this case, the spaces for both the plain text and cipher text is K^n . So the public key is a map from K^n to K^n . It consists of $n + 1$ quartic polynomials over n variables $(x_1, x_2, \dots, x_n)$ and the instruction to evaluate those polynomials using the plain text and then divide the result of each of the first n evaluations by the evaluation of the $(n + 1)^{st}$ polynomial. The coefficients in these polynomials are all from k , a subfield of K . The resulting function is comprised of n components, where each is a map defined by a rational expression over n variables using coefficients in k .

With the map as defined, specifically using coefficients only from k , the public key serves as a map from k^n to k^n as well as from K^n to K^n . This construction adds to the efficiency of RMPKC by enabling the public key to use coefficients from k , a smaller field, and encrypt messages from K^n , a vector space over a larger field. This use of the two separate fields has no bearing on the cryptanalysis of RMPKC, therefore throughout this presentation we will consider only one field, and call it k .

The private key is the means to invert the map of the public key. Several maps are joined together by composition to form the public key, and the means to invert these maps, put together, provide a way to invert the whole public key, thus serving as the private key. Our mission in the cryptanalysis is to start with a public key and define a private key that inverts it.

3.1 The public key

The public key as defined by the authors is given as a set of rational degree 4 functions:

$$\left(\frac{P_1(x_1, \dots, x_n)}{P_{n+1}(x_1, \dots, x_n)}, \dots, \frac{P_n(x_1, \dots, x_n)}{P_{n+1}(x_1, \dots, x_n)} \right),$$

where each $P_i : k^n \longrightarrow k$ is defined by a quartic polynomial over k in n variables. In this paper, we will separate the given public key into two separate maps: a division map and a polynomial map. Let us define the division map ϕ as follows:

$$\phi : k^{n+1} \longrightarrow k^n, \quad \phi(x_1, x_2, \dots, x_{n+1}) = \begin{pmatrix} \frac{x_1}{x_{n+1}} \\ \frac{x_2}{x_{n+1}} \\ \vdots \\ \frac{x_n}{x_{n+1}} \end{pmatrix}, \quad x_{n+1} \neq 0,$$

and the polynomial map P as

$$P : k^n \longrightarrow k^{n+1}, P = \begin{pmatrix} P_1 \\ P_2 \\ \vdots \\ P_{n+1} \end{pmatrix}, P(x_1, x_2, \dots, x_n) = \begin{pmatrix} P_1(x_1, x_2, \dots, x_n) \\ P_2(x_1, x_2, \dots, x_n) \\ \vdots \\ P_{n+1}(x_1, x_2, \dots, x_n) \end{pmatrix}$$

where for relevant values of i , P_i is as defined above. Naturally then, the public key is defined by $\phi \circ P$.

This public key is constructed as the composition of the following:

$$\phi \circ P = L_3 \circ \phi \circ G \circ L_2 \circ \phi \circ F \circ L_1$$

where

$$k^n \xrightarrow{L_1} k^n \xrightarrow{F} k^{n+1} \xrightarrow{\phi} k^n \xrightarrow{L_2} k^n \xrightarrow{G} k^{n+1} \xrightarrow{\phi} k^n \xrightarrow{L_3} k^n.$$

Here L_1, L_2, L_3 are homogeneous, multivariate linear maps. For $i = 1, 2, 3$,

$$L_i : k^n \longrightarrow k^n, \quad L_i = \begin{pmatrix} L_{i,1} \\ L_{i,2} \\ \vdots \\ L_{i,n} \end{pmatrix}$$

where for $1 \leq j \leq n$, $L_{i,j}$ is defined by a homogeneous, multivariate linear polynomial:

$$L_{i,j}(x_1, x_2, \dots, x_n) = \sum_{r=1}^n l_{i,j[r]} x_r.$$

For all relevant values of (i, j, r) , $l_{i,j[r]}$ is a constant, randomly chosen from k . The random choices for the coefficients $l_{i,j[r]}$ will not necessarily result in an invertible map L_i . In the case that they do not, those random choices get scrapped. Rechoosing values for the coefficients continues until a selection is found that does produce a fully invertible map L_i .

Thus L_1, L_2 , and L_3 are invertible maps. We will sometimes refer to them as linear transformations.

Both F and G are quadratic polynomial maps, i.e. each consists of $n + 1$ quadratic polynomial functions, $k^n \rightarrow k$.

$$F = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{n+1} \end{pmatrix}, \quad G = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_{n+1} \end{pmatrix}$$

where for $1 \leq i \leq n + 1$, f_i and g_i are quadratic polynomials in $(x_1, \dots, x_n)$ as described in the previous chapter on notation.

The details of the construction of F and G are provided below in the section explaining the private key. F and G are constructed identically, with different choices of random parameters.

It may not be obvious why the composition of these seven maps, of which two are division maps, yields such a simple result of a public key with $n + 1$ quartic polynomials and the instruction to divide each of the first n by the last. We will not provide a justification at this point. As part of the first step of the cryptanalysis, we analyze closely the composition of the public key and create other formulations which show why the simple result is obtained.

Along with the public key polynomial map P , the RMPKC owner must provide the field and its operations $(K, +, *)$ for public use. With the above information in hand, anyone can encrypt a message using the RMPKC.

3.2 Encryption

Given a plaintext $X' = (x'_1, \dots, x'_n) \in K^n$ the public user computes the ciphertext $Y' = (y'_1, \dots, y'_n) \in K^n$ by $Y' = \phi \circ P(X')$, meaning

$$\begin{pmatrix} y'_1 \\ y'_2 \\ \vdots \\ y'_n \end{pmatrix} = \begin{pmatrix} \frac{p_1(x'_1, x'_2, \dots, x'_n)}{p_{n+1}(x'_1, x'_2, \dots, x'_n)} \\ \frac{p_2(x'_1, x'_2, \dots, x'_n)}{p_{n+1}(x'_1, x'_2, \dots, x'_n)} \\ \vdots \\ \frac{p_n(x'_1, x'_2, \dots, x'_n)}{p_{n+1}(x'_1, x'_2, \dots, x'_n)} \end{pmatrix}$$

The result, $(y'_1, \dots, y'_n)$, is the ciphertext, which is sent to the RMPKC owner, who holds the private key, for decryption. Ideally, the RMPKC owner maintains secret this private key. (Unfortunately, for RMPKC, as we show in this thesis, a third party may develop a private key based on the public key functions published by the owner.)

3.3 The private key

The private key is the means to invert the maps which together compose the public key. The elegant structure of RMPKC includes an easy way to invert the maps $\phi \circ F$ and $\phi \circ G$. So when thinking about the public key we like to visualize

$$k^n \xrightarrow{P} k^{n+1} \xrightarrow{\phi} k^n$$

and when thinking about the private key we like to visualize

$$\begin{array}{ccccccc} L_1 & & \phi \circ F & & L_2 & & \phi \circ G & & L_3 \\ k^n \rightarrow k^n & \longrightarrow & k^n \rightarrow k^n & \longrightarrow & k^n \rightarrow k^n & \longrightarrow & k^n \rightarrow k^n \end{array}$$

The invertible linear transformations L_1, L_2 , and L_3 have computible inverses, L_1^{-1}, L_2^{-1} , and L_3^{-1} . These would be computed once and stored as part of the private key.

However, the maps $\phi \circ G$ and $\phi \circ F$ are not fully invertible, so their inverses, per se, do not exist. Nonetheless, as we shall show, a unique pre-image of each of G and F does exist in almost every case. We will explain precisely how to obtain the pre-image in the section below entitled "Decryption".

The authors of RMPKC conceptualized what we call $\phi \circ F$ as one map. So here we will explain the structure of F by explaining how the authors structured $\phi \circ F$ (the same applies to $\phi \circ G$). $\phi \circ F$ contains two distinct parts: a core transformation and a sequential part. The core transformation maps the last two components of the domain space, which we'll call x_{n-1} and x_n , to the last two components of the image space, which we'll call y_{n-1} and y_n . The sequential part uses all n components of the domain space to map to the first $n - 2$ components of the image space.

$$\begin{pmatrix} x_1 \\ \\ x_2 \\ \vdots \\ x_{n-2} \\ \hline x_{n-1} \\ \\ x_n \end{pmatrix} \longrightarrow \begin{pmatrix} \frac{f_1(x_1, \dots, x_n)}{f_{n+1}(x_{n-1}, x_n)} \\ \\ \frac{f_2(x_2, \dots, x_n)}{f_{n+1}(x_{n-1}, x_n)} \\ \vdots \\ \frac{f_{n-2}(x_{n-2}, \dots, x_n)}{f_{n+1}(x_{n-1}, x_n)} \\ \hline \frac{f_{n-1}(x_{n-1}, x_n)}{f_{n+1}(x_{n-1}, x_n)} \\ \\ \frac{f_n(x_{n-1}, x_n)}{f_{n+1}(x_{n-1}, x_n)} \end{pmatrix}$$

In the diagram, the sequential part lays above the core transformation. Note carefully the subscripts on the function parameters. There is a clear pattern in the sequential part, which we'll explain below.

The mathematics behind the core transformation is more complex, yielding a simple and elegant result. The mathematics behind the sequential part is quite simple. We will explain first the core transformation, since in the decryption process, it must be inverted first.

The creators found a way to define the core transformation as a quadratic, rational function with two components that has a closed form expression for its "inverse" function. This inverse is a linear rational function with two components, as we will show shortly.

To construct the polynomials f_{n-1} , f_n , and f_{n+1} , which comprise the core transformation, we randomly choose 12 elements in k : $\alpha_1, \dots, \alpha_6$ and $\beta_1, \dots, \beta_6$. f_{n-1} , f_n and f_{n+1} are defined as follows:

$$f_{n-1}(x_{n-1}, x_n) = \tau_{n-1,1}x_{n-1}x_n + \tau_{n-1,2}x_{n-1} + \tau_{n-1,3}x_n + \tau_{n-1,4}$$

$$f_n(x_{n-1}, x_n) = \tau_{n,1}x_{n-1}x_n + \tau_{n,2}x_{n-1} + \tau_{n,3}x_n + \tau_{n,4}$$

$$f_{n+1}(x_{n-1}, x_n) = \tau_{n+1,1}x_{n-1}x_n + \tau_{n+1,2}x_{n-1} + \tau_{n+1,3}x_n + \tau_{n+1,4}$$

where the $\tau_{i,j}$ is defined as follows:

$$\tau_{n-1,1} = \alpha_6\beta_4 - \alpha_4\beta_6 \quad \tau_{n,1} = \alpha_5\beta_6 - \alpha_6\beta_5 \quad \tau_{n+1,1} = \alpha_4\beta_5 - \alpha_5\beta_4$$

$$\tau_{n-1,2} = \alpha_1\beta_6 - \alpha_3\beta_4 \quad \tau_{n,2} = \alpha_3\beta_5 - \alpha_2\beta_6 \quad \tau_{n+1,2} = \alpha_2\beta_4 - \alpha_1\beta_5$$

$$\tau_{n-1,3} = \alpha_4\beta_3 - \alpha_6\beta_1 \quad \tau_{n,3} = \alpha_6\beta_2 - \alpha_5\beta_3 \quad \tau_{n+1,3} = \alpha_5\beta_1 - \alpha_4\beta_2$$

$$\tau_{n-1,4} = \alpha_3\beta_1 - \alpha_1\beta_3 \quad \tau_{n,4} = \alpha_2\beta_3 - \alpha_3\beta_2 \quad \tau_{n+1,4} = \alpha_1\beta_2 - \alpha_2\beta_1$$

The inverse of the core transformation is given by:

$$(x_{n-1}, x_n) = \left(\frac{\beta_1 y_n + \beta_2 y_{n-1} + \beta_3}{\beta_4 y_n + \beta_5 y_{n-1} + \beta_6}, \frac{\alpha_1 y_n + \alpha_2 y_{n-1} + \alpha_3}{\alpha_4 y_n + \alpha_5 y_{n-1} + \alpha_6} \right).$$

The elegance of this structure provides for a very simple private key and for very fast decryption. To find the pre-image of the core transformation of F , the RMPKC owner needs only to store the six α 's and six β 's.

The remaining portion of F is the sequential part, built in triangular form. Its construction is simple:

$$\forall 1 \leq i \leq n-2, f_i(x_i, \dots, x_n) = a_i(x_{i+1}, \dots, x_n)x_i + b_i(x_{i+1}, \dots, x_n),$$

where the a_i 's are randomly chosen linear polynomials and the b_i 's are randomly chosen quadratic polynomials. This construction enables the following fact:

$\forall i$, given that the values for the variables $x_{i+1}, \dots, x_n$ are known, f_i is linear in x_i .

This fact makes the process of finding the pre-image of the sequential part of F very easy, as we shall show below.

3.4 Decryption

Recall from the section above on encryption, a plaintext $X' = (x'_1, \dots, x'_n)$ was transformed into the ciphertext $Y' = (y'_1, \dots, y'_n)$ by $Y' = \phi \circ P(X')$.

To decrypt, we are given Y' and need to find X' such that $Y' = \phi \circ P(X')$. The holder of the private key has the means to find the inverse of each of $L_3, (\phi \circ G), L_2, (\phi \circ F), L_1$. Performing the calculations in order will yield $(x'_1, \dots, x'_n)$. Inversion of the linear transformations L_3, L_2 , and L_1 is obvious. Inversion of each of $\phi \circ F$ and $\phi \circ G$ requires some attention.

To invert the map $\phi \circ F$ is to find the solution of the equation:

$$\phi \circ F(x_1, x_2, \dots, x_n) = \begin{pmatrix} y'_1 \\ y'_2 \\ \vdots \\ y'_n \end{pmatrix}$$

when given $y'_1, y'_2, \dots, y'_n \in K$. As explained above in the section entitled "The Private Key", $\phi \circ F$ is comprised of two distinct parts: the core transformation and the sequential part. In decryption, we first invert the core transformation.

Recall that the core transformation has a closed form for its inverse:

$$(x_{n-1}, x_n) = \left(\frac{\beta_1 y_n + \beta_2 y_{n-1} + \beta_3}{\beta_4 y_n + \beta_5 y_{n-1} + \beta_6}, \frac{\alpha_1 y_n + \alpha_2 y_{n-1} + \alpha_3}{\alpha_4 y_n + \alpha_5 y_{n-1} + \alpha_6} \right).$$

By inserting values y'_{n-1} and y'_n for variables y_{n-1} and y_n respectively, we find solutions for x_{n-1} and x_n , which we shall call x'_{n-1} and x'_n . Once values for x_{n-1} and x_n are found, we use the polynomial expressions of the sequential part, together with the evaluation of the denominator polynomial, to solve, in order, for $x_{n-2}, x_{n-3}, \dots, x_1$.

For $i = n-2, n-3, \dots, 1$, we shall solve for x_i given $\forall i < j \leq n, x_j = x'_j$ and given known values $y'_1, y'_2, \dots, y'_n$. First recall that the polynomial expression for the i^{th} component

of F is

$$f_i(x_1, x_2, \dots, x_n) = a_i(x_{i+1}, x_{i+2}, \dots, x_n) * x_i + b_i(x_{i+1}, x_{i+2}, \dots, x_n)$$

where a_i is a linear polynomial and b_i is a quadratic polynomial. So in decryption we get:

$$y'_i = \frac{f_i(x_1, x_2, \dots, x_n)}{f_{n+1}(x_1, x_2, \dots, x_n)} = \frac{a_i(x_{i+1}, x_{i+2}, \dots, x_n) * x_i + b_i(x_{i+1}, x_{i+2}, \dots, x_n)}{(\tau_{n-2,1}x_{n-1}x'_n + \tau_{n-2,2}x_{n-1} + \tau_{n-2,3}x_n + \tau_{n-2,4})}$$

where we have for $j = i + 1, i + 2, \dots, n, x_j = x'_j$, known values, y'_i is a known value, and each subscripted τ is a known constant (part of the private key). Applying these known values gives us

$$y'_i = \frac{a_i(x'_{i+1}, x'_{i+2}, \dots, x'_n) * x_i + b_i(x'_{i+1}, x'_{i+2}, \dots, x'_n)}{(\tau_{n-2,1}x'_{n-1}x'_n + \tau_{n-2,2}x'_{n-1} + \tau_{n-2,3}x'_n + \tau_{n-2,4})}.$$

We can solve for x_i , yielding

$$x_i = \frac{y'_i * (\tau_{n-2,1}x'_{n-1}x'_n + \tau_{n-2,2}x'_{n-1} + \tau_{n-2,3}x'_n + \tau_{n-2,4}) - b_i(x'_{i+1}, x'_{i+2}, \dots, x'_n)}{a_i(x'_{i+1}, x'_{i+2}, \dots, x'_n)}.$$

Note that we also have formulae for each subscripted polynomial a and b ; therefore, we are able to perform the calculation and obtain a value for x_i , which we'll naturally name x'_i . By applying this method sequentially for $i = n - 2, n - 3, \dots, 1$, we can find a complete pre-image for $\phi \circ F$.

Inversion of $\phi \circ G$ is performed in the exact same manner as $\phi \circ F$: first the core transformation, then the sequential part, component by component.

So one can see that the decryption of RMPKC is quite simple and extremely fast.

3.5 Division by Zero

The public key of RMPKC, $\phi \circ P$ is not fully well-defined. Neither is the decryption process, which we'll call $(\phi \circ P)^{-1}$. The linear transformations L_1, L_2 and L_3 are fully invertible and thus create no concerns here. However, rational expressions are present in

$\phi \circ P$ and division is required to invert $\phi \circ F$ and $\phi \circ G$. The division functions involved are vulnerable to the problem of division by zero.

In the encryption process, the only divisor is $P_{n+1}(x_1, \dots, x_n)$. In the decryption process, there are n different divisors required to invert each of $\phi \circ F$ and $\phi \circ G$. The two divisors in each of the core transformations are of the form

$$\alpha_4 y_n + \alpha_5 y_{n-1} + \alpha_6, \quad \beta_4 y_n + \beta_5 y_{n-1} + \beta_6.$$

The $n - 2$ divisors in the sequential parts in each of F and G are in the form

$$a_i(x_i, x_{i+1}, \dots, x_n), \quad (\forall 1 \leq i \leq n - 1)$$

where a_i is a randomly chosen linear polynomial.

Given an RMPKC public key generated as described, and given a randomly chosen element from K^n as message text, the probability of division by zero in the encryption process is $\frac{1}{|K|}$.

To estimate the probability of encountering division by zero during the process of decryption, consider independently the probability of encountering division by zero in each of the $2n$ division calculations performed in decryption. In these cases, the estimated probability of division by zero is again $\frac{1}{|K|}$. So the estimated probability of encountering division by zero at any time during the decryption process is

$$\begin{aligned} & 2\left[\frac{1}{|K|} + \left(1 - \frac{1}{|K|}\right)\frac{1}{|K|} + \left(1 - \frac{1}{|K|}\right)^2\frac{1}{|K|} + \dots + \left(1 - \frac{1}{|K|}\right)^{n-1}\frac{1}{|K|}\right] \\ &= \frac{2}{|K|} \sum_{r=0}^{n-1} \left(1 - \frac{1}{|K|}\right)^r \\ &= \frac{2}{|K|} \sum_{r=0}^{n-1} \left(\frac{|K|-1}{|K|}\right)^r < \frac{2n}{|K|}. \end{aligned}$$

Typically, $n \ll |K|$ so this probability is very small.

3.6 Memory and Computational Requirements

The advantage of potentially increased security afforded by using a public key with quartic polynomials, rather than quadratic polynomials, is countered by the disadvantages of requiring a larger public key and of requiring more computations in encryption and decryption. In this section we will explore these requirements.

The public key of RMPKC consists of $n + 1$ polynomials of degree four in n variables. That means that the total number of coefficients from k that must be stored as the public key is

$$(n + 1) \binom{(n + 1) + 3}{4}.$$

As one can see, the public key size is $O(n^5)$. The following table shows the actual size in bytes for cryptosystems of various sizes through $n = 25$.

n	terms per polynomial	terms per public key	bytes per public key $ k = 2^{16}$	bytes per public key $ k = 2^{32}$
6	210	1,470	2,940	5,880
7	330	2,640	5,280	10,560
8	495	4,455	8,910	17,820
9	715	7,150	14,300	28,600
10	1,001	11,011	22,022	44,044
15	3,876	62,016	124,032	248,064
20	10,626	223,146	446,292	892,584
25	23,751	617,526	1,235,052	2,470,104

The computations required to execute encryption and decryption are also important

considerations for any cryptosystem. The standard measure used is the number of multiplications required. We will also count divisions, since RMPKC differs from other similar cryptosystems by the use of division. For RMPKC, with $n + 1$ non-homogeneous quartic polynomials in the public key, one of which being the common denominator for all n rational expressions, we require

$$n \left[4 \binom{n+3}{4} + 3 \binom{n+2}{3} + 2 \binom{n+1}{2} + \binom{n}{1} \right] = \frac{n^2}{6} [n^3 + 9n^2 + 26n + 24]$$

multiplications and n divisions for encryption.

For decryption, we need to perform three linear transformations of the size n by n , invert two core transformations, and invert two sequential portions of the functions F and G . Each of the linear transformations requires n^2 multiplications. Each core transformation inversion requires eight multiplications and two divisions. The inversion of each sequential portion requires

$$\sum_{r=2}^n 1 + 2 \binom{r}{2} + r + r = \frac{1}{2} [n^2 + 3n - 6] \sum_{r=1}^n r^2$$

multiplications and $n - 2$ divisions. The total number of multiplications for decryption, therefore, is

$$3[n^2] + 2[8] + 2 \left\{ \frac{1}{2} [n^2 + 3n - 6] \sum_{r=1}^n r^2 \right\} = 3n^2 + 16 + [n^2 + 3n - 6] \sum_{r=1}^n r^2,$$

and the number of divisions required is

$$2[2] + 2[n - 2] = 2n.$$

Once can see that RMPKC is designed extremely well, especially for decryption.

3.7 Small-Scale Example

The small-scale example we present here provides the reader a chance to more clearly see how RMPKC works. We use $|k| = 11$ and $n = 3$ to make a cryptosystem rather manageable

for comparison by hand calculations. We also chose simplified versions of the linear transformations, the alpha and beta coefficient vectors, and for the quadratic polynomials in the sequential solution set.

$$L_1(x_1, x_2, x_3) = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \quad L_2(x_1, x_2, x_3) = \begin{pmatrix} x_1 \\ x_1 + x_2 \\ x_3 \end{pmatrix}, \quad L_3(x_1, x_2, x_3) = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

For F , we have

$$\alpha = (0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1), \quad \beta = (0 \quad 2 \quad 0 \quad 1 \quad 0 \quad 0)$$

$$f_1(x_1, x_2, x_3) = x_1 x_2$$

$$f_2(x_1, x_2, x_3) = x_2 x_3$$

$$f_3(x_1, x_2, x_3) = 2x_3$$

$$f_4(x_1, x_2, x_3) = x_2.$$

For G , we have

$$\delta = (0 \quad 3 \quad 0 \quad 0 \quad 0 \quad 1), \quad \gamma = (0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0)$$

$$g_1(x_1, x_2, x_3) = 2x_1 x_2 + x_1 x_3 + 3x_1$$

$$g_2(x_1, x_2, x_3) = x_2 x_3$$

$$g_3(x_1, x_2, x_3) = x_3$$

$$g_4(x_1, x_2, x_3) = 3x_2.$$

The following are formulations of the public key which we will carefully define in the following chapter. for now, please accept the formulations, which ease the construction of the public key via composition of the individual maps.

$$\phi \circ P = L_3 \circ \phi \circ G \circ L_2 \circ \phi \circ F \circ L_1$$

$$\phi \circ P = \phi \circ \tilde{L}_3 \circ G \circ \phi \circ \tilde{L}_2 \circ F \circ L_1$$

$$\phi \circ P = \phi \circ \tilde{L}_3 \circ \tilde{G} \circ \tilde{L}_2 \circ F \circ L_1.$$

For our small-scale example with identity matrices for each of L_1 and L_3 we have

$$\phi \circ P = \phi \circ \tilde{G} \circ \tilde{L}_2 \circ F$$

where

$$\tilde{L}_2(x_1, x_2, x_3, x_4) = \begin{pmatrix} x_1 \\ x_1 + x_2 \\ x_3 \\ x_4 \end{pmatrix}, \tilde{G}(x_1, x_2, x_3, x_4) = \begin{pmatrix} 2x_1x_2 + x_1x_3 + 3x_1x_4 \\ x_2x_3 \\ x_3x_4 \\ 3x_2x_4 \end{pmatrix}.$$

Now we combine the maps to form the public key:

$$\tilde{L}_2 \circ F(x_1, x_2, x_3) = \begin{pmatrix} x_1x_2 \\ x_1x_2 + x_2x_3 \\ 2x_3 \\ x_2 \end{pmatrix}$$

$$P = \tilde{G} \circ L_2 \circ F = \begin{pmatrix} 2x_1^2x_2^2 + 2x_1x_2^2x_3 + 2x_1x_2x_3 + 3x_1x_2^2 \\ 2x_1x_2x_3 + 2x_2x_3^2 \\ 2x_2x_3 \\ 3x_1x_2^2 + 3x_2^2x_3 \end{pmatrix}$$

The private key requires, in addition to the information given above, the inverse of L_2 ,

namely $L_2^{-1}(x_1, x_2, x_3) = \begin{pmatrix} x_1 \\ 10x_1 + x_2 \\ x_3 \end{pmatrix}.$

For this toy example, we will encrypt $X = (6 \ 5 \ 3).$

$$P(6, 5, 3) = \begin{pmatrix} 2(6)^2(5)^2 + 2(6)(5)^2(3) + 2(6)(5)(3) + 3(6)(5)^2 \\ 2(6)(5)(3) + 2(5)(3)^2 \\ 2(5)(3) \\ 3(6)(5)^2 + 3(5)^2(3) \end{pmatrix}$$

$$= \begin{pmatrix} 2(3)(3) + 3(3) + 4 + 7(3) \\ 4 + 2 \\ 8 \\ 7(3) + 9(3) \end{pmatrix} = \begin{pmatrix} 8 \\ 6 \\ 8 \\ 4 \end{pmatrix}$$

$$\phi \circ P(6, 5, 3) = \begin{pmatrix} \frac{8}{4} \\ \frac{6}{4} \\ \frac{8}{4} \end{pmatrix} = \begin{pmatrix} 2 \\ 7 \\ 2 \end{pmatrix}$$

Now to decrypt $Y = \begin{pmatrix} 2 \\ 7 \\ 2 \end{pmatrix}$, we start by applying the inverse of the core transforma-

tion of G Recall that $L_3(x_1, x_2, x_3) = (L_3)^{-1}(x_1, x_2, x_3) = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$. Since $(\gamma_1, \gamma_2, \dots, \gamma_6) = (0, 1, 0, 1, 0, 0)$ and $(\delta_1, \delta_2, \dots, \delta_6) = (0, 3, 0, 0, 0, 1)$,

$$\begin{pmatrix} 2 \\ 7 \\ 2 \end{pmatrix} \longrightarrow \begin{pmatrix} \frac{\gamma_1(2)+\gamma_2(7)+\gamma_3}{\gamma_4(2)+\gamma_5(7)+\gamma_6} \\ \frac{\delta_1(2)+\delta_2(7)+\delta_3}{\delta_4(2)+\delta_5(7)+\delta_6} \end{pmatrix} = \begin{pmatrix} \frac{(0)(2)+(1)(7)+(0)}{(1)(2)+(0)(7)+(0)} \\ \frac{(0)(2)+(3)(7)+(0)}{(0)(2)+(0)(7)+(1)} \end{pmatrix} = \begin{pmatrix} \frac{7}{2} \\ \frac{10}{1} \end{pmatrix} = \begin{pmatrix} 9 \\ 10 \end{pmatrix}$$

Now for the sequential part of G ,

$$\frac{g_1(x_1, x_2, x_3)}{g_4(x_1, x_2, x_3)} = \frac{2x_1x_2 + x_1x_3 + 3x_1}{3x_2} = 2$$

$$\frac{g_1(x_1, 9, 10)}{g_4(x_1, 9, 10)} = 2 = \frac{2x_1(9) + x_1(10) + 3x_1}{3(9)} = \frac{10x_1}{8}$$

yields $x_1 = \frac{5}{10} = 6$. Thus we get $Y = \begin{pmatrix} 2 \\ 7 \\ 2 \end{pmatrix} \longrightarrow \begin{pmatrix} 6 \\ 9 \\ 10 \end{pmatrix}$ per applying the inversion of

$$\phi \circ G \text{ to } \begin{pmatrix} 2 \\ 7 \\ 2 \end{pmatrix}.$$

We next apply the L_2^{-1} to this results, yielding

$$L_2^{-1}(6, 9, 10) = \begin{pmatrix} 6 \\ 10(6) + 9 \\ 10 \end{pmatrix} = \begin{pmatrix} 6 \\ 3 \\ 10 \end{pmatrix}.$$

Finally, we get to F . We start with the inverse of the core transformation:

$$(\beta_1, \beta_2, \dots, \beta_6) = (0, 2, 0, 1, 0, 0), \quad (\alpha_1, \alpha_2, \dots, \alpha_6) = (0, 1, 0, 0, 0, 1)$$

$$\begin{pmatrix} 6 \\ 3 \\ 10 \end{pmatrix} \longrightarrow \begin{pmatrix} \frac{\beta'_1(10) + \beta'_2(3) + \beta'_3}{\beta'_4(10) + \beta'_5(3) + \beta'_6} \\ \frac{\alpha'_1(10) + \alpha'_2(3) + \alpha'_3}{\alpha'_4(10) + \alpha'_5(3) + \alpha'_6} \end{pmatrix} = \begin{pmatrix} \frac{(0)(10) + (2)(3) + (0)}{(1)(10) + (0)(3) + (0)} \\ \frac{(0)(10) + (1)(3) + (0)}{(0)(10) + (0)(3) + (1)} \end{pmatrix} = \begin{pmatrix} \frac{6}{10} \\ \frac{3}{1} \end{pmatrix} = \begin{pmatrix} 5 \\ 3 \end{pmatrix}$$

At lastly, we address the sequential part of F ,

$$\frac{f_1(x_1, x_2, x_3)}{f_4(x_1, x_2, x_3)} = 6 = \frac{x_1 x_2}{x_2} = x_1 = 6$$

Thus,

$$\begin{pmatrix} 6 \\ 3 \\ 10 \end{pmatrix} \longrightarrow \begin{pmatrix} 6 \\ 5 \\ 3 \end{pmatrix},$$

which is the original plain text. Our encryption and decryption processes were successful.

This same toy example will be used in chapter 5 to demonstrate our cryptanalysis.

Chapter 4

2R

In the following chapter, we will explain in full the cryptanalysis of RMPKC. One of the key components of our cryptanalysis is an application of another cryptanalytic approach previously published, known as the cryptanalysis of 2R. In this chapter, we first describe the cryptographic system called 2R. Then we describe the cryptanalysis of that system as published by Ye, Lam, and Dai. In the following chapter, we will explain our attack including a detailed explanation of how we expand on the work of Ye, Lam, and Dai in their attack on 2R.

4.1 Cryptographic System 2R

In the mid 1990's, as part of the earlier effort to investigate the possibility of building on Matsumoto and Imai's early polynomial-based multivariate public key cryptosystem, J. Patarin and L. Goubin developed the idea of composing non-linear maps as the key part of a cryptographic scheme [7] [8]. Patarin and Goubin's idea was based on the realization that all of the so-called one round schemes were vulnerable to attack. The one round scheme was a public key based on the composition of only one non-linear function with linear functions. The linear functions hide the structure of the non-linear function in composition.

So the two-round schemes, abbreviated 2R, were developed to decrease the vulnerability of the cryptosystem by forming a public key which is constructed by a composition of two

one-round schemes. The aim of this composition is to hide the structures of the non-linear functions of each of the one-round schemes. The thought was that the composition of two non-linear functions would be much more effective than the composition of a non-linear function with linear functions.

The structures of the non-linear functions of the public key in the two-round schemes are definitely related to the vulnerability of the two-round scheme to attack. Ye, Lam, and Dai list five appropriate structures for the non-linear functions.

1. Monomials over an extension field of degree n of the form $a \rightarrow a^{1+|k|\theta}$ for some constant θ ;
2. Triangular functions;
3. S-box functions;
4. Combinations of triangular functions with S-boxes.
5. D^* functions, which are comprised of squaring functions over an extension field of degree n . These work when $|k|^n \equiv -1 \pmod{4}$.

Of course since Ye, Lam, and Dai published their attack in 1999, numerous other non-linear structures for functions have been discovered and worked on. Notably HFE (Hidden Field Equations) is a generalization of the first item. The specific structures employed in the cryptosystems do make a difference in the effectiveness of the attack on 2R. The authors claim that cryptosystems based on the composition of non-linear functions in the form of the first two structures are weak, citing other work by Patarin and Goubin. They then focus on a scheme called D^{**} , a special case of D^* . In D^{**} , $|k|^n \equiv -1 \pmod{4}$, the non-linear functions are squaring functions over an extension field of degree n , and the message block space is restricted to be such that when the public key acts on that space, it is injective.

The added restriction is irrelevant to the attack, so we think about their focus as if it were on D^* .

Now although Ye, Lam, and Dai successfully attacked D^{**} [24], they did not explain the application of their attack on a wider variety of 2R structures. We have found out that this attack on 2R, as described by Ye, Lam, and Dai, is insufficient when applied to RMPKC. The reason for this does not stem directly from the fact that RMPKC uses for its non-linear maps rational functions rather than polynomial functions. It actually comes from the construction of the non-linear functions. RMPKC uses a combination of the triangular maps with the core transformation, which is the 2x2 map described previously. Both of the portions of the non-linear maps (sequential solution [i.e. triangular] and the core transformation) create the need for additional work necessary to apply Ye, Lam, and Dai’s attack to RMPKC.

The authors also glossed over the issue of homogeneity of the quadratic polynomials used in the construction of the quartics. For now, we will gloss over the same issue, but address it later when we apply this approach to RMPKC, which does not employ homogeneous polynomials.

4.2 Cryptanalysis of 2R

The attack on 2R by Ye, Lam, and Dai is focused on breaking down quartic polynomials into their component quadratics. We say ‘breaking down’ because the quartic polynomials under consideration were formed specifically by composing quadratic polynomials.

The insight that Ye, Lam, and Dai provide is substantial. The process required to employ the attack on 2R is simple. It requires four steps [24].

1. Take partial derivatives of each of the quartic polynomials with respect to each of the $n + 1$ variables;
2. Recover the span of the polynomials of the first quadratic function in the first compo-

nent non-linear function;

3. Choose a basis from this space;
4. Identify the polynomials that when composed with the basis found in #4 above form the given quartic polynomials.

Note that the notation in this chapter deviates somewhat from the more consistent notation used in the rest of this presentation. Choosing the letters g and f to represent the quadratic polynomials in question was done on purpose of the obvious analogy between 2R and RMPKC.

Let g and f each be a set of n quadratic homogenous polynomials in n variables.

$$g = \begin{pmatrix} g_1(y_1, y_2, \dots, y_n) \\ g_1(y_1, y_2, \dots, y_n) \\ \vdots \\ g_n(y_1, y_2, \dots, y_n) \end{pmatrix}, \quad f = \begin{pmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) \end{pmatrix}$$

Let $g \circ f$ be the set of quartic polynomials formed by the composition of the given quadratic polynomials.

$$g \circ f = \begin{pmatrix} g_1 \circ f(x_1, x_2, \dots, x_n) \\ g_2 \circ f(x_1, x_2, \dots, x_n) \\ \vdots \\ g_n \circ f(x_1, x_2, \dots, x_n) \end{pmatrix}$$

We take partial derivatives of each of the composed quartics $g_1, \dots, g_{n+1}$. Recall from calculus,

$$\frac{\partial}{\partial x_j}(g_i \circ f) = \sum_{r=1}^n \frac{\partial}{\partial w_r} g_i(w_1, \dots, w_n) \times \frac{\partial}{\partial x_j} f_r(x_1, \dots, x_n)$$

where $w_1, \dots, w_n$ are dummy variables representing, respectively, $f_1, \dots, f_n$ in the composition of the multivariate polynomials.

Let's take a close look at the above representation of $\frac{\partial}{\partial x_j}(g_i \circ f)$. It is merely a summation of terms, where each term is a product of two expressions. The first expression, $\frac{\partial}{\partial w_r} g_i(w_1, \dots, w_n)$, is linear in the $w_1, \dots, w_n$'s because $g_i(w_1, \dots, w_n)$ is quadratic in the $w_1, \dots, w_n$'s.

The second expression, $\frac{\partial}{\partial x_j} f_r(x_1, \dots, x_n)$ ($\forall r$) is linear in the $x_1, \dots, x_n$'s also because $f_r(x_1, \dots, x_n)$ is quadratic in the $x_1, \dots, x_n$'s.

Since both g and f are homogeneous, and $(w_1, \dots, w_n)$ represent, respectively, $f_1, \dots, f_n$, the product of the first and second expression is in the form $x_i f_j(x_1, \dots, x_n)$ for some $1 \leq j \leq n$ (multiplied by some constant). This results is important because it causes

$D \subseteq \lambda$, where

$$D = \text{Span}\left\{ \frac{\partial}{\partial x_j} g_i \circ f(x_1, \dots, x_n) : 1 \leq i, j \leq n \right\} \text{ and}$$

$$\lambda = \text{Span}\left\{ x_i f_j(x_1, \dots, x_n) : 1 \leq i, j \leq n \right\}.$$

To recover $\mathcal{F} = \text{Span}\{ f_r(x_1, \dots, x_n) : 1 \leq r \leq n \}$, we find

$$R = \{ \theta : \forall 1 \leq i \leq n, x_i \theta \in D \}.$$

If $D = \lambda$ then $R = \mathcal{F}$. The argument for $D = \lambda$ being true in most cases is that for randomly generated f 's, the set $\{ \frac{\partial}{\partial x_j} g_i \circ f(x_1, \dots, x_n) : 1 \leq i, j \leq n+1 \}$ is linearly independent. We do not have a proof for this, but our empirical tests showed it true for a wide range of n . The authors of Cryptanalysis of 2R offer heuristic arguments. Note that with the f 's of RMPKC, which are not randomly generated but rather follow a specific structure with several random coefficients, $D = \lambda$ is never true. The cryptanalysts of 2R do not develop a technique for when $D \subset \lambda$; we develop a technique specifically for RMPKC, which will be described later.

D is generated from the given quartic polynomials. The dimension of D can be calculated.

As long as the dimension of D is n^2 , we know $D = \lambda$. Recovering R from D is easy in this case.

With $R = \mathcal{F}$, the work is almost done. We need a basis for this space and some linear algebra to complete the job.

Let $\bar{f}_1, \dots, \bar{f}_n$ be a basis for R . We will find $\bar{g}_1, \dots, \bar{g}_n$ such that

$$\forall k, \bar{g}_k \circ \begin{pmatrix} \bar{f}_1(x_1, x_2, \dots, x_n) \\ \bar{f}_2(x_1, x_2, \dots, x_n) \\ \vdots \\ \bar{f}_n(x_1, x_2, \dots, x_n) \end{pmatrix} = g_k \circ \begin{pmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) \end{pmatrix}$$

To accomplish this, define

$$\forall k, \bar{g}_k(x_1, \dots, x_n) = \sum_{i=1}^n \sum_{j=1}^n a_{k,i,j} x_i x_j.$$

We need to find the $a_{k,i,j}$'s. Since the $\bar{f}$'s are known, we can substitute them for the variables to form:

$$\bar{g}_k \circ \begin{pmatrix} \bar{f}_1(x_1, x_2, \dots, x_n) \\ \bar{f}_2(x_1, x_2, \dots, x_n) \\ \vdots \\ \bar{f}_n(x_1, x_2, \dots, x_n) \end{pmatrix} = \sum_{i=1}^n \sum_{j=1}^n a_{k,i,j} f_i(x_1, x_2, \dots, x_n) f_j(x_1, x_2, \dots, x_n)$$

and set the resulting expression equal to

$$g_k \circ \begin{pmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) \end{pmatrix}$$

which we have (it is one of the original quartic polynomials which we are given). So $\forall k$, we have one equality. But this one equality has a quartic polynomial on either side. For

the quartic polynomials to be equal, each and every coefficient must be equal. So we have effectively t equalities, where t represents the number of terms in the quartic polynomial.

Note

$$t = \binom{n}{4} + n \binom{n-1}{2} + \binom{n}{2} + 2 \binom{n}{2} + n, \quad \forall n > 3$$

and t is greater than the number of unknowns $\binom{n}{2}$. Furthermore, each of the t inequalities are linear in the $a_{k,i,j}$'s, making finding the solution easy.

We have found neither of the following exact component functions:

$$\begin{pmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) \end{pmatrix}, \quad \begin{pmatrix} g_1(x_1, x_2, \dots, x_n) \\ g_2(x_1, x_2, \dots, x_n) \\ \vdots \\ g_n(x_1, x_2, \dots, x_n) \end{pmatrix}.$$

However, for some linear transformation, L , we do get the following:

$$\begin{pmatrix} \bar{f}_1(x_1, x_2, \dots, x_n) \\ \bar{f}_2(x_1, x_2, \dots, x_n) \\ \vdots \\ \bar{f}_n(x_1, x_2, \dots, x_n) \end{pmatrix} = L \circ \begin{pmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) \end{pmatrix}.$$

Furthermore, since

$$\begin{aligned} \forall k, g_k \circ \begin{pmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) \end{pmatrix} &= \bar{g}_k \circ \begin{pmatrix} \bar{f}_1(x_1, x_2, \dots, x_n) \\ \bar{f}_2(x_1, x_2, \dots, x_n) \\ \vdots \\ \bar{f}_n(x_1, x_2, \dots, x_n) \end{pmatrix}, \\ \bar{g}_k \circ \begin{pmatrix} \bar{f}_1(x_1, x_2, \dots, x_n) \\ \bar{f}_2(x_1, x_2, \dots, x_n) \\ \vdots \\ \bar{f}_n(x_1, x_2, \dots, x_n) \end{pmatrix} &= \bar{g}_k \circ \left(L \circ \begin{pmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) \end{pmatrix} \right) \end{aligned}$$

$$= (\bar{g}_k \circ L) \circ \begin{pmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) \end{pmatrix}.$$

Thus $g_k = \bar{g}_k \circ L$.

So we have not found the exact quadratic polynomials that were used to form the given quartic polynomials, but we have found quadratic polynomials that when combined via composition, do form the same quartic polynomial. Finding such polynomials is the most we could accomplish. The following explains why.

Given the original set of component homogeneous quadratic polynomials g and f , and given any invertible linear transformation L , consider

$$g \circ L \circ L^{-1} \circ f.$$

$g \circ L$ is a set of component homogeneous quadratic polynomials; and $L^{-1} \circ f$ is also a set of component homogeneous quadratic polynomials. As one can see,

$$g \circ L \circ L^{-1} \circ f = g \circ f.$$

This completes our explanation of the attack on 2R. Now we will shift our attention to our attack on RMPKC.

Chapter 5

Cryptanalysis of RMPKC

5.1 Overview

Our cryptanalysis of RMPKC takes a public key from an RMPKC and produces a private key for that system. The previous chapter explained a cryptanalysis of 2R by Ye, Lam, and Dai. 2R is a cryptographic system based on multivariate quartic polynomial maps. RMPKC is a cryptographic system based on multivariate quartic rational maps. In this overview, we give a brief explanation of the six steps used in our attack. Following this brief explanation, we provide an overview using mathematical notation.

Following the overview, we will explain in detail how and why each step works. A separate section will be dedicated to each step of the attack. The mathematical challenges mentioned here in the overview will be explained in the following sections.

The first step of our attack is to understand how the attack on polynomial maps can be applied to the polynomials that form the rational maps and to prep these polynomials for the attack. Seeing how to apply the attack is only a mathematical exercise, with no computations required. Preparation of the polynomials is performed simply by homogenization.

The second step of our attack is to apply the partial derivative attack, described in the previous chapter (attacking 2R), on RMPKC. The structure of the quadratic polynomials used in RMPKC to form the public key makes it impossible for the attack on 2R to work

directly on RMPKC. A most significant part of our research is to enhance this attack, enabling it to successfully work on RMPKC. The result of the application of this attack on RMPKC is a list of multivariate quadratic polynomials, which in part form the given quartic polynomials of the given public key.

Specifically, the multivariate quadratic polynomials found in the second step form the first part of the composition scheme. A second set of multivariate quadratic polynomials are required to complete the formation of the given multivariate quartic polynomial maps under composition. Furthermore, the multivariate quadratic polynomials found in the second step of our attack are not easy to invert. They are likely very different from the multivariate quadratic polynomials originally used to construct the public key.

Thus, the third step of our attack is to decompose the set of multivariate quadratic polynomials found in the second step into a combination of a linear transformation and another set of multivariate quadratic polynomials. When combined under function composition, the linear transformation and multivariate quadratic polynomial map are equivalent to the map consisting of the multivariate quadratic polynomials resulting from the second step.

Furthermore, the third step specifically chooses the linear transformation and the second set of multivariate quadratic polynomials such that the multivariate quadratic polynomials are structured in the exact same way as the multivariate quadratic polynomials used in the formation of an RMPKC public key. Our attack could have omitted the third step, and left as an additional exercise the inversion of the multivariate quadratic polynomials from the second step. Including the third step serves two purposes: 1) makes the attack strong, by fully forming an alternate private key; and 2) by forcing more learning about the structure of the multivariate quadratic polynomial maps used in RMPKC.

The fourth step of our attack identifies the ordered set of multivariate quadratic polynomials, which combined with the multivariate quadratic polynomial map from step 2 under

composition, form the multivariate quartic polynomial map of the public key. This step is by far the easiest of the six steps, requiring only commonly known algebraic techniques. However, the multivariate quadratic polynomials identified in the fourth step are also not easily invertible.

Thus, the fifth step, similarly to the third step, is to find linear transformations (two in this case) and a multivariate quadratic polynomial map which together (combined via composition of functions) are equivalent to the map consisting of the multivariate quadratic polynomials found in the fourth step. The procedure employed in the fifth step is quite similar to that used in the third step.

The sixth and last step is merely an adjustment to the results of step 5 which get the last of the linear transformations into the necessary form.

Now we give an overview of our attack, using mathematical notation and a slightly more mathematical explanation. Recall we start with $n + 1$ multivariate quartic polynomials. They are ordered 1 through n , with the last one ($n + 1$), being the designated denominator polynomial. Let ϕ be the division map and let P be the multivariate quartic polynomial map as follows:

$$\phi : k^{n+1} \longrightarrow k^n, \phi(x_1, \dots, x_{n+1}) = \left(\frac{x_1}{x_{n+1}}, \frac{x_2}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}} \right),$$

$$P : k^n \longrightarrow k^{n+1}, P(x_1, \dots, x_n) = \begin{pmatrix} p_1(x_1, \dots, x_n) \\ p_2(x_1, \dots, x_n) \\ \vdots \\ p_{n+1}(x_1, \dots, x_n) \end{pmatrix},$$

where $\forall 1 \leq i \leq n + 1$, p_i is a multivariate quartic polynomial.

Execution of the public key (converting plain text to cipher text) requires $\phi \circ P : k^n \longrightarrow k^n$, but the public key, which is available to the cryptanalyst, includes the multivariate

quartic polynomials of P , namely

$$p_1(x_1, x_2, \dots, x_n), p_2(x_1, x_2, \dots, x_n), \dots, p_{n+1}(x_1, x_2, \dots, x_n).$$

Recall from chapter 2 (The RMPKC Cryptosystem), the formation of the multivariate quartic polynomial map of the public key, namely P :

$$\phi \circ P = L_3 \circ \phi \circ G \circ L_2 \circ \phi \circ F \circ L_1,$$

where L_1, L_2 , and L_3 are linear transformations from k^n to k^n , ϕ is the division function with the common denominator as described previously, and

$$G, F : k^n \longrightarrow k^{n+1},$$

$$G(x_1, \dots, x_n) = \begin{pmatrix} g_1(x_1, \dots, x_n) \\ g_2(x_1, \dots, x_n) \\ \vdots \\ g_{n+1}(x_1, \dots, x_n) \end{pmatrix}, F(x_1, \dots, x_n) = \begin{pmatrix} f_1(x_1, \dots, x_n) \\ f_2(x_1, \dots, x_n) \\ \vdots \\ f_{n+1}(x_1, \dots, x_n) \end{pmatrix}$$

where $\forall 1 \leq i \leq n+1$, $g_i(x_1, \dots, x_n)$ and $f_i(x_1, \dots, x_n)$ are multivariate quadratic polynomials.

Our attack will ultimately yield a similar composition of the public key, namely

$$\phi \circ P = L'_3 \circ \phi \circ G' \circ L'_2 \circ \phi \circ F' \circ L'_1$$

where each map in this composition, which includes a $(')$, is analogous to its counterpart in the original composition without the $(')$. During the construction of this private key, we are careful to construct each component with the exact same structure as done in RMPKC so that our new composition actually serves as a private key.

In the first step of our attack, we show that the composition of maps which were used originally to make P , can be seen differently.

Our understanding of these composite maps of P will lead us to another formulation for the public key, namely the following:

$$\phi \circ P = \phi \circ \hat{G} \circ \bar{F},$$

where

$$\hat{G} : k^{n+1} \longrightarrow k^{n+1}, \hat{G}(x_1, \dots, x_{n+1}) = \begin{pmatrix} \hat{g}_1(x_1, \dots, x_{n+1}) \\ \hat{g}_2(x_1, \dots, x_{n+1}) \\ \vdots \\ \hat{g}_{n+1}(x_1, \dots, x_{n+1}) \end{pmatrix},$$

where $\forall 1 \leq i \leq n+1$, $\hat{g}_i(x_1, \dots, x_n)$ is a multivariate homogenous quadratic polynomial, and

$$\bar{F} : k^n \longrightarrow k^{n+1}, \bar{F}(x_1, \dots, x_n) = \begin{pmatrix} \bar{f}_1(x_1, \dots, x_n) \\ \bar{f}_2(x_1, \dots, x_n) \\ \vdots \\ \bar{f}_{n+1}(x_1, \dots, x_n) \end{pmatrix}$$

where $\forall 1 \leq i \leq n+1$, $\bar{f}_i(x_1, \dots, x_n)$ is a multivariate quadratic polynomial (not necessarily homogeneous).

The preparation of the public key for the partial derivative attack is accomplished by homogenizing the multivariate quartic polynomials of P . We will denote the homogenized version by $\tilde{P}$. This preparation yields the following:

$$\phi \circ \tilde{P} = \phi \circ \hat{G} \circ \hat{F},$$

where ϕ is as defined previously and

$$\hat{G}, \hat{F} : k^{n+1} \longrightarrow k^{n+1},$$

$$\hat{G}(x_1, \dots, x_{n+1}) = \begin{pmatrix} \hat{g}_1(x_1, \dots, x_{n+1}) \\ \hat{g}_2(x_1, \dots, x_{n+1}) \\ \vdots \\ \hat{g}_{n+1}(x_1, \dots, x_{n+1}) \end{pmatrix}, \hat{F}(x_1, \dots, x_{n+1}) = \begin{pmatrix} \hat{f}_1(x_1, \dots, x_{n+1}) \\ \hat{f}_2(x_1, \dots, x_{n+1}) \\ \vdots \\ \hat{f}_{n+1}(x_1, \dots, x_{n+1}) \end{pmatrix}$$

where $\forall 1 \leq i \leq n+1$, $\hat{g}_i(x_1, \dots, x_{n+1})$ and $\hat{f}_i(x_1, \dots, x_{n+1})$ are multivariate homogenous quadratic polynomials. This homogenization of the public key marks the completion of the first step of the attack.

The second step of the attack takes the multivariate homogenized quartic polynomials of the public key, namely $\tilde{p}_1(x_1, \dots, x_{n+1}), \tilde{p}_2(x_1, \dots, x_{n+1}), \dots, \tilde{p}_{n+1}(x_1, \dots, x_{n+1})$, and applies the partial derivative attack to yield

$$\mathcal{F}_H = \text{Span} \{ \tilde{f}_i(x_1, x_2, \dots, x_{n+1}) : 1 \leq i \leq n+1 \}.$$

The partial derivative attack forms the following sets:

$$D = \text{Span} \left\{ \frac{\partial}{\partial x_j} [\tilde{p}_i(x_1, \dots, x_{n+1})] : 1 \leq i, j \leq n+1 \right\}$$

$$R = \{ \theta : \forall 1 \leq i \leq n+1, x_i \theta \in D \}.$$

If all goes well, which it does for Ye, Lam, and Dai in their cryptanalysis of 2R, $R = \mathcal{F}_H$. Unfortunately, in the case of RMPKC, $R \subset \mathcal{F}_H$ but $R \neq \mathcal{F}_H$.

Define

$$\Lambda = \{ x_j \theta : 1 \leq j \leq m, \theta \in \mathcal{F}_H \}$$

to help explain what is going on here. We will show and prove some relations among the relevant spaces.

In the details section, we will define for $s = 1, 2, \dots, n+1$, D_s , a set analogous to D but with $x_s = 0$. We also define R_s as the following:

$$R_s = \{ \theta : \forall 1 \leq i \leq n+1, x_i \theta \in D_s \}.$$

We build from R_s to create R_s^+ . Then we make $R^+ = \cap_{s=1}^{n+1} R_s^+$. We succeed in obtaining $R^+ = \mathcal{F}_H$. The complexity of the definition of these sets analogous to D and R increases with increasing values for n . We find success with this method for several values of n on the order of what the original cryptographers proposed for RMPKC, but we have theoretical proof neither for why it works, seemingly every time, nor that it would work for much larger values of n .

The second step is concluded by de-homogenizing $\mathcal{F}_H$ to form $\mathcal{F}$. The homogenized form of the public key is necessary only to find $\mathcal{F}$. The rest of our attack will be based on the original form of the public key, namely $\phi \circ P$.

The third step of our attack focuses only on the set

$$\mathcal{F} = \text{Span} \{ f_i(x_1, x_2, \dots, x_n) : 1 \leq i \leq n+1 \}.$$

The second step has given us $\mathcal{F}$, but not the specific multivariate quadratic polynomials $f_i(x_1, x_2, \dots, x_n) \forall 1 \leq i \leq n+1$. We will not find these specific polynomials, but we will carefully form a basis for $\mathcal{F}$ which enables us to form an alternate private key for the RMPKC. The result of this third step will be a linear transformation, L'_1 , and a multivariate quadratic polynomial map F' as follows:

$$L'_1 : k^n \longrightarrow k^n,$$

$$F' : k^n \longrightarrow k^{n+1}, F'(x_1, \dots, x_n) = \begin{pmatrix} f'_1(x_1, \dots, x_n) \\ f'_2(x_1, \dots, x_n) \\ \vdots \\ f'_{n+1}(x_1, \dots, x_n) \end{pmatrix}$$

where $\forall 1 \leq i \leq n+1$, $f'_i(x_1, \dots, x_n)$ is a multivariate quadratic polynomial. Furthermore,

$$\text{Span} \{ f'_i \circ L_1(x_1, x_2, \dots, x_n) : 1 \leq i \leq n+1 \} = \mathcal{F}$$

and

$$f_{n+1} \circ L_1(x_1, x_2, \dots, x_n) \in \text{Span} \{ f'_{n+1} \circ L_1(x_1, x_2, \dots, x_n) \}.$$

Note that in this context, $f'_i \circ L_1(x_1, x_2, \dots, x_n)$ represents the polynomial formed by composing the multivariate linear polynomials which comprise L_1 (defined in the most obvious way) with the multivariate quadratic polynomial $f'_i(x_1, x_2, \dots, x_n)$.

Note also that $f_{n+1} \circ L_1(x_1, x_2, \dots, x_n) \in \text{Span} \{ f'_{n+1} \circ L_1(x_1, x_2, \dots, x_n) \}$ is important because f_{n+1} is the denominator of the rational expressions forming the map F . A multiplicative constant relating f'_{n+1} to f_{n+1} has no bearing on the cryptanalysis.

Additionally, the multivariate quadratic polynomials $f'_i(x_1, x_2, \dots, x_n)$, $\forall 1 \leq i \leq n+1$ form the map F' which legitimately fits into the RMPKC structure in the same manner as the multivariate quadratic polynomial map F , which was used to create the system. We construct these multivariate quadratic polynomials $f'_i(x_1, x_2, \dots, x_n)$ in such a way so that we can use the RMPKC inversion process on them. The linear transformation L'_1 , the multivariate quadratic polynomial map F' , and the means to invert F' become part of the alternate private key, which is the end goal of our cryptanalysis.

The fourth step of our process finds the multivariate quadratic polynomial map $\hat{G}'$, where

$$\hat{G}' : k^{n+1} \longrightarrow k^{n+1}, \hat{G}'(x_1, x_2, \dots, x_{n+1}) = \begin{pmatrix} \hat{g}'_1(x_1, x_2, \dots, x_{n+1}), \\ \hat{g}'_2(x_1, x_2, \dots, x_{n+1}), \\ \vdots \\ \hat{g}'_{n+1}(x_1, x_2, \dots, x_{n+1}) \end{pmatrix}$$

which, when composed with the maps found in the third step, namely L'_1 and F' , form

the map

$$P = \hat{G}' \circ F' \circ L'_1$$

Specifically, we obtain

$$\forall 1 \leq i \leq n+1, p_i(x_1, x_2, \dots, x_n) = \hat{g}'_i(f' \circ L_1(x_1, x_2, \dots, x_n)).$$

Basic algebraic techniques, which will be detailed in the appropriate following section, are all that are required to complete this fourth step.

The fifth step works in much the same way as the third step. We start with

$$\hat{G}' = \begin{pmatrix} \hat{g}'_1(x_1, x_2, \dots, x_{n+1}) \\ \hat{g}'_2(x_1, x_2, \dots, x_{n+1}) \\ \vdots \\ \hat{g}'_{n+1}(x_1, x_2, \dots, x_{n+1}) \end{pmatrix}$$

where $\phi \circ P = \phi \circ \hat{G}' \circ F' \circ L'_1$.

We do not know how to invert $\hat{G}'$, so we will find a linear transformations L'_2 and a multivariate quadratic polynomial map $\bar{G}''$ such that the span of the component polynomials of $\hat{G}'$ is the same as the span of the component polynomials of $\bar{G}'' \circ L'_2$. An additional linear transformation, which we'll call $\hat{L}'_3$ will be necessary in order to complete the picture:

$$\phi \circ P = \phi \circ \hat{G}' \circ F' \circ L'_1 = \phi \circ \hat{L}'_3 \circ \bar{G}'' \circ L'_2 \circ \phi \circ F' \circ L'_1$$

,

where

$$L'_2 : k^n \longrightarrow k^n,$$

$$\hat{L}'_3 : k^{n+1} \longrightarrow k^{n+1},$$

$$\hat{G}'' : k^n \longrightarrow k^{n+1}, \quad \hat{G}''(x_1, x_2, \dots, x_n) = \begin{pmatrix} \hat{g}_1''(x_1, x_2, \dots, x_n) \\ \hat{g}_2''(x_1, x_2, \dots, x_n) \\ \vdots \\ \hat{g}_{n+1}''(x_1, x_2, \dots, x_n) \end{pmatrix}$$

Additionally the multivariate quadratic polynomials $\hat{g}_i''(x_1, x_2, \dots, x_n)$, $\forall 1 \leq i \leq n+1$ form the map $\hat{G}''$ which legitimately fits into the RMPKC structure in the same manner as the multivariate quadratic polynomial map G , which was used to create the system. We construct these multivariate quadratic polynomials $\hat{g}_i''(x_1, x_2, \dots, x_n)$ in such a way so that we can use the RMPKC inversion process on them.

The sixth and final step will first find appropriate $\tilde{L}'_3$ and G' such that $\tilde{L}'_3 \circ G' = \hat{L}'_3 \circ \bar{G}''$. Then we will define L'_3 such that $\phi \circ \tilde{L}'_3 \circ G' = L'_3 \circ \phi \circ G'$. Additionally in step 6 we find the means to invert G' .

This discussion concludes the overview of our cryptanalysis on RMPKC. The following six sections will cover in detail each step of the attack. But before that, we now summarize the mathematics used in our six-step attack:

1. Step 1: Re-formulate the Public Key and Homogenize

Reformulate the public key:

$$\phi \circ P = L_3 \circ \phi \circ G \circ L_2 \circ \phi \circ F \circ L_1$$

$$\phi \circ P = L_3 \circ \phi \circ G \circ \phi \circ \tilde{L}_2 \circ F \circ L_1$$

$$\phi \circ P = L_3 \circ \phi \circ \tilde{G} \circ \tilde{L}_2 \circ F \circ L_1$$

$$\phi \circ P = \phi \circ \tilde{L}_3 \circ \tilde{G} \circ \tilde{L}_2 \circ F \circ L_1$$

$$\phi \circ P = \phi \circ \hat{G} \circ F \circ L_1$$

$$\phi \circ P = \phi \circ \hat{G} \circ \bar{F}$$

Homogenize the public key:

$$\phi \circ \tilde{P} = \phi \circ \hat{G} \circ \hat{F}$$

2. Step 2: Find $\mathcal{F}$

$$\mathcal{F} = Span \{ \bar{f}_i \}$$

3. Step 3: Find F' and L'_1

$$\mathcal{F} = Span \{ f'_i \circ L'_1 \} = Span \{ \bar{f}_i \}$$

4. Step 4: Find $\bar{G}'$

$$P = \hat{G}' \circ F' \circ L'_1$$

$$P = \bar{G}' \circ \phi \circ F' \circ L'_1$$

5. Step 5: Find $\bar{G}''$ and L'_2

$$P = \hat{L}'_3 \circ \bar{G}'' \circ L'_2 \circ \phi \circ F' \circ L'_1$$

6. Step 6: Find G' and L'_3

$$P = \tilde{L}'_3 \circ G' \circ L'_2 \circ \phi \circ F' \circ L'_1$$

$$\phi \circ P = L'_3 \circ \phi \circ G' \circ L'_2 \circ \phi \circ F' \circ L'_1$$

5.2 Step 1: Homogenize the Public Key

Our first step's objective is to prepare a given public key of RMPKC for the partial derivative attack, which was explained in chapter 2. We achieve this objective in two mini-steps. First, we study the formulation of the public key and restate it in a way that enables the application directly of the partial derivative attack on the multivariate quartic polynomials that form the numerators and common denominator of the public key, except for the fact that the polynomials are not homogeneous.

Then we homogenize the very same multivariate quartic polynomials, making them fully prepared for the partial derivative attack. Recall that RMPKC ends up being invulnerable to the direct application of the partial derivative attack, and that we will need to adapt the attack to succeed on RMPKC. This adaptation will be fully explained in the following section covering step 2.

First, we address the formulation of the public key. Recall that the public key $\phi \circ P$ is formed by the composition of the following maps:

$$\phi \circ P = L_3 \circ \phi \circ G \circ L_2 \circ \phi \circ F \circ L_1,$$

$$\text{where } P : k^n \longrightarrow k^{n+1}, P(x_1, x_2, \dots, x_n) = \begin{pmatrix} p_1(x_1, x_2, \dots, x_n) \\ p_2(x_1, x_2, \dots, x_n) \\ \vdots \\ p_{n+1}(x_1, x_2, \dots, x_n) \end{pmatrix}, \forall 1 \leq i \leq n+1, p_i$$

is a quartic polynomial, L_1, L_2 , and L_3 are linear transformations from k^n to k^n , ϕ is the division function

$$\phi : k^{n+1} \longrightarrow k^n, \phi(x_1, x_2, \dots, x_{n+1}) = \begin{pmatrix} x_1/x_{n+1} \\ x_2/x_{n+1} \\ \vdots \\ x_n/x_{n+1} \end{pmatrix},$$

and $G, F : k^n \longrightarrow k^{n+1}$,

$$G(x_1, \dots, x_n) = \begin{pmatrix} g_1(x_1, \dots, x_n) \\ g_2(x_1, \dots, x_n) \\ \vdots \\ g_{n+1}(x_1, \dots, x_n) \end{pmatrix}, F(x_1, \dots, x_n) = \begin{pmatrix} f_1(x_1, \dots, x_n) \\ f_2(x_1, \dots, x_n) \\ \vdots \\ f_{n+1}(x_1, \dots, x_n) \end{pmatrix}$$

where $\forall 1 \leq i \leq n+1$, $g_i(x_1, \dots, x_n)$ and $f_i(x_1, \dots, x_n)$ are quadratic polynomials. Recall that these quadratic polynomials are purposely structured in a way to render the inversion of each of F and G easy. We will lose this nice structure during the partial derivative attack applied in step 2; then we will work in step 3 to re-create the nice structure for our version of F , namely F' , and we will work in step 5 to re-create the nice structure for our version of G , namely G' .

Let's define linear transformation $\tilde{L}_2$ as follows:

$$\tilde{L}_2 : k^{n+1} \longrightarrow k^{n+1}, \tilde{L}_2(x_1, x_2, \dots, x_{n+1}) = \begin{pmatrix} L_2(x_1, x_2, \dots, x_n) \\ \dots\dots\dots \\ x_{n+1} \end{pmatrix}.$$

It is not hard to show that $L_2 \circ \phi = \phi \circ \tilde{L}_2$. Thus

$$\phi \circ P = L_3 \circ \phi \circ G \circ L_2 \circ \phi \circ F \circ L_1 = L_3 \circ \phi \circ G \circ \phi \circ \tilde{L}_2 \circ F \circ L_1.$$

Now consider $G \circ \phi$.

$$G \circ \phi : k^{n+1} \longrightarrow k^{n+1}, G \circ \phi(x_1, x_2, \dots, x_{n+1}) = \begin{pmatrix} g_1(\phi(x_1, x_2, \dots, x_{n+1})) \\ g_2(\phi(x_1, x_2, \dots, x_{n+1})) \\ \vdots \\ g_{n+1}(\phi(x_1, x_2, \dots, x_{n+1})) \end{pmatrix}$$

$$= \begin{pmatrix} g_1 \left(\frac{x_1}{x_{n+1}}, \frac{x_2}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}} \right) \\ g_2 \left(\frac{x_1}{x_{n+1}}, \frac{x_2}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}} \right) \\ \vdots \\ g_{n+1} \left(\frac{x_1}{x_{n+1}}, \frac{x_2}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}} \right) \end{pmatrix}$$

Since each g_i is a multivariate quadratic polynomial of n variables, $\forall 1 \leq i \leq n+1$,

$$g_i(x_1, x_2, \dots, x_{n+1}) = \sum_{j=1}^n \sum_{k=j}^n g_{i[j,k]} x_j x_k + \sum_{j=1}^n g_{i[j,n+1]} x_j + g_{i[n+1,n+1]}$$

for some $\{ g_{i[j,k]} : 1 \leq i \leq n+1, 1 \leq j \leq k \leq n+1 \}$. Thus $\forall 1 \leq i \leq n+1$,

$$\begin{aligned} g_i \left(\frac{x_1}{x_{n+1}}, \frac{x_2}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}} \right) &= \sum_{j=1}^{n+1} \sum_{k=j}^{n+1} g_{i[j,k]} \frac{x_j}{x_{n+1}} \frac{x_k}{x_{n+1}} + \sum_{j=1}^n g_{i[j,n+1]} \frac{x_j}{x_{n+1}} + g_{i[n+1,n+1]} \\ &= \sum_{j=1}^{n+1} \sum_{k=j}^{n+1} g_{i[j,k]} \frac{x_j}{x_{n+1}} \frac{x_k}{x_{n+1}} + \sum_{j=1}^n g_{i[j,n+1]} \frac{x_j}{x_{n+1}} \frac{x_{n+1}}{x_{n+1}} + g_{i[n+1,n+1]} \frac{x_{n+1}}{x_{n+1}} \frac{x_{n+1}}{x_{n+1}} \end{aligned}$$

In this last step, multiplication of some of the terms by $\frac{x_{n+1}}{x_{n+1}}$ is legitimate because $x_{n+1} \neq 0$; otherwise ϕ would not be defined. Because of the clever choice of subscripts, we can combine all of the terms into the following single summation

$$= \sum_{j=1}^{n+1} \sum_{k=j}^{n+1} g_{i[j,k]} \frac{x_j}{x_{n+1}} \frac{x_k}{x_{n+1}}$$

Finally, we factor out the denominator which does not vary over the terms to yield

$$= \frac{1}{x_{n+1}^2} \left[\sum_{j=1}^{n+1} \sum_{k=j}^{n+1} g_{i[j,k]} x_j x_k \right].$$

$\forall 1 \leq i \leq n+1$ let $\tilde{g}_i$ be defined as the homogenized version of g_i , namely

$$\tilde{g}_i(x_1, x_2, \dots, x_{n+1}) = \sum_{j=1}^{n+1} \sum_{k=j}^{n+1} g_{i[j,k]} x_j x_k$$

where the set $\{ g_{i[j,k]} : 1 \leq i \leq n+1, 1 \leq j \leq k \leq n+1 \}$ is the same as described above, in connection with the polynomials denoted g_i , $\forall 1 \leq i \leq n+1$.

We have just shown the following:

$$G \circ \phi : k^{n+1} \longrightarrow k^{n+1}, G \circ \phi(x_1, x_2, \dots, x_{n+1}) = \begin{pmatrix} \frac{1}{x_{n+1}^2} \tilde{g}_1(x_1, x_2, \dots, x_{n+1}) \\ \frac{1}{x_{n+1}^2} \tilde{g}_2(x_1, x_2, \dots, x_{n+1}) \\ \vdots \\ \frac{1}{x_{n+1}^2} \tilde{g}_{n+1}(x_1, x_2, \dots, x_{n+1}) \end{pmatrix}$$

Now consider $\phi \circ G \circ \phi$.

$$\phi \circ G \circ \phi : k^{n+1} \longrightarrow k^n,$$

$$\phi \circ G \circ \phi(x_1, x_2, \dots, x_{n+1}) = \begin{pmatrix} \frac{1}{x_{n+1}^2} \tilde{g}_1(x_1, x_2, \dots, x_{n+1}) / \frac{1}{x_{n+1}^2} \tilde{g}_{n+1}(x_1, x_2, \dots, x_{n+1}) \\ \frac{1}{x_{n+1}^2} \tilde{g}_2(x_1, x_2, \dots, x_{n+1}) / \frac{1}{x_{n+1}^2} \tilde{g}_{n+1}(x_1, x_2, \dots, x_{n+1}) \\ \vdots \\ \frac{1}{x_{n+1}^2} \tilde{g}_n(x_1, x_2, \dots, x_{n+1}) / \frac{1}{x_{n+1}^2} \tilde{g}_{n+1}(x_1, x_2, \dots, x_{n+1}) \end{pmatrix}$$

$$\phi \circ G \circ \phi(x_1, x_2, \dots, x_{n+1}) = \begin{pmatrix} \tilde{g}_1(x_1, x_2, \dots, x_{n+1}) / \tilde{g}_{n+1}(x_1, x_2, \dots, x_{n+1}) \\ \tilde{g}_2(x_1, x_2, \dots, x_{n+1}) / \tilde{g}_{n+1}(x_1, x_2, \dots, x_{n+1}) \\ \vdots \\ \tilde{g}_n(x_1, x_2, \dots, x_{n+1}) / \tilde{g}_{n+1}(x_1, x_2, \dots, x_{n+1}) \end{pmatrix}$$

So we define $\tilde{G}$ as the following map:

$$\tilde{G} : k^{n+1} \longrightarrow k^{n+1}, \tilde{G}(x_1, x_2, \dots, x_{n+1}) = \begin{pmatrix} \tilde{g}_1(x_1, x_2, \dots, x_{n+1}) \\ \tilde{g}_2(x_1, x_2, \dots, x_{n+1}) \\ \vdots \\ \tilde{g}_{n+1}(x_1, x_2, \dots, x_{n+1}) \end{pmatrix}$$

where one can see that $\phi \circ \tilde{G} = \phi \circ G \circ \phi$, which gives us

$$\phi \circ P = L_3 \circ \phi \circ G \circ \phi \circ \tilde{L}_2 \circ F \circ L_1 = L_3 \circ \phi \circ \tilde{G} \circ \tilde{L}_2 \circ F \circ L_1.$$

Let's define linear transformation $\tilde{L}_3$ in a manner similarly to how we defined $\tilde{L}_2$, namely

$$\tilde{L}_3 : k^{n+1} \longrightarrow k^{n+1}, \tilde{L}_3(x_1, x_2, \dots, x_{n+1}) = \begin{pmatrix} L_3(x_1, x_2, \dots, x_n) \\ \dots\dots\dots \\ x_{n+1} \end{pmatrix}.$$

Again we have $L_3 \circ \phi = \phi \circ \tilde{L}_3$. Thus

$$\phi \circ P = \phi \circ \tilde{L}_3 \circ \hat{G} \circ \tilde{L}_2 \circ F \circ L_1.$$

Let's define $\hat{G} = \tilde{L}_3 \circ \tilde{G}$ and $\bar{F} = \tilde{L}_2 \circ F \circ L_1$. $\tilde{L}_3$, $\tilde{L}_2$, and L_1 are linear transformations,

$$\hat{G} : k^{n+1} \longrightarrow k^{n+1}, \hat{G}(x_1, x_2, \dots, x_{n+1}) = \begin{pmatrix} \hat{g}_1(x_1, x_2, \dots, x_{n+1}) \\ \hat{g}_2(x_1, x_2, \dots, x_{n+1}) \\ \vdots \\ \hat{g}_{n+1}(x_1, x_2, \dots, x_{n+1}) \end{pmatrix}$$

$$\bar{F} : k^n \longrightarrow k^{n+1}, \bar{F}(x_1, x_2, \dots, x_n) = \begin{pmatrix} \bar{f}_1(x_1, x_2, \dots, x_n) \\ \bar{f}_2(x_1, x_2, \dots, x_n) \\ \vdots \\ \bar{f}_{n+1}(x_1, x_2, \dots, x_n) \end{pmatrix}$$

where $\forall 1 \leq i \leq n+1$, $\hat{g}_i$ and $\bar{f}_i$ are quadratic polynomials. Furthermore, $\forall 1 \leq i \leq n+1$, $\hat{g}_i$ is homogeneous. Thus our public key can be expressed as

$$\phi \circ P = \phi \circ \hat{G} \circ \bar{F}$$

It is true that $P = \hat{G} \circ \bar{F}$ even though in this explanation we dropped the multiplier $\frac{1}{x_{n+1}^2}$

from the definition of $\hat{G}$, so it stayed out of the definition of $\hat{G}$. The reason why $P = \hat{G} \circ \bar{F}$ is true is that in P , the same multiplier is dropped out. Including the multiplier in the public

key quartic polynomial list would not change the key but would unnecessarily provide some additional information to cryptanalysts.

Thus the $n + 1$ multivariate quartic polynomials $p_1, p_2, \dots, p_{n+1}$ provided as part of the public key are exactly the composition of the homogeneous quadratic polynomials $\hat{g}_1, \hat{g}_2, \dots, \hat{g}_{n+1}$ with the quadratic polynomials $\bar{f}_1, \bar{f}_2, \dots, \bar{f}_{n+1}$ as follows: $\forall 1 \leq i \leq n + 1$,

$$p_i(x_1, x_2, \dots, x_n) = \hat{g}_i \left(\bar{f}_1(x_1, x_2, \dots, x_n), \bar{f}_2(x_1, x_2, \dots, x_n), \dots, \bar{f}_{n+1}(x_1, x_2, \dots, x_n) \right).$$

The partial derivative attack works on multivariate homogeneous quartic polynomials (those formed by composing multivariate homogeneous quadratic polynomials). So we will not use this attack directly on the polynomials of P , but rather on the homogenized version of the same polynomials. Define the following:

$$\forall 1 \leq i \leq n + 1, \tilde{p}_i(x_1, x_2, \dots, x_{n+1}) = x_{n+1}^4 p_i \left(\frac{x_1}{x_{n+1}}, \frac{x_2}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}} \right).$$

Note that $\forall 1 \leq i \leq n + 1$, $\tilde{p}_i(x_1, x_2, \dots, x_n, 1) = p_i(x_1, x_2, \dots, x_n)$. The homogenization of the polynomials of P homogenize the polynomials $\hat{G} \circ \bar{F}$ by homogenizing the polynomials $\bar{f}_1, \bar{f}_2, \dots, \bar{f}_{n+1}$. We show this truth as follows. Since $\forall 1 \leq i \leq n + 1$, $\hat{g}_i(x_1, x_2, \dots, x_{n+1})$ is homogeneous, $\forall 1 \leq i \leq n + 1$,

$$\hat{g}_i(x_1, x_2, \dots, x_{n+1}) = \sum_{j=1}^{n+1} \sum_{k=j}^{n+1} \hat{g}_{i[j,k]} x_j x_k$$

for some set $\{\hat{g}_{i[j,k]} : 1 \leq i \leq n + 1, 1 \leq j \leq k \leq n + 1\}$. Thus $\forall 1 \leq i \leq n + 1$,

$$\begin{aligned} \hat{g}_i \left(\bar{f}_1(x_1, x_2, \dots, x_n), \bar{f}_2(x_1, x_2, \dots, x_n), \dots, \bar{f}_{n+1}(x_1, x_2, \dots, x_n) \right) \\ = \sum_{j=1}^{n+1} \sum_{k=j}^{n+1} \hat{g}_{i[j,k]} \bar{f}_j(x_1, x_2, \dots, x_n), \bar{f}_k(x_1, x_2, \dots, x_n) \end{aligned}$$

$\forall 1 \leq i \leq n + 1$, we have $\tilde{p}_i(x_1, x_2, \dots, x_{n+1}) =$

$$x_{n+1}^4 \sum_{j=1}^{n+1} \sum_{k=j}^{n+1} \hat{g}_{i[j,k]} \bar{f}_j \left(\frac{x_1}{x_{n+1}}, \frac{x_2}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}} \right) \bar{f}_k \left(\frac{x_1}{x_{n+1}}, \frac{x_2}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}} \right)$$

$$= \sum_{j=1}^{n+1} \sum_{k=j}^{n+1} \hat{g}_{i[j,k]} \left[x_{n+1}^2 \bar{f}_j \left(\frac{x_1}{x_{n+1}}, \frac{x_2}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}} \right) \right] \left[x_{n+1}^2 \bar{f}_k \left(\frac{x_1}{x_{n+1}}, \frac{x_2}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}} \right) \right].$$

The homogenization of $\bar{f}_j$ and $\bar{f}_k$ are exactly as you see in the above expression, namely

$$\left[x_{n+1}^2 \bar{f}_j \left(\frac{x_1}{x_{n+1}}, \frac{x_2}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}} \right) \right], \quad \left[x_{n+1}^2 \bar{f}_k \left(\frac{x_1}{x_{n+1}}, \frac{x_2}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}} \right) \right].$$

This concludes our discussion of step one. We started with the public key $(\phi \circ P)$ composed as it is by three linear transformations (L_3, L_2, L_1) and two multivariate non-homogeneous quadratic rational maps (G, F) as follows:

$$\phi \circ P = L_3 \circ \phi \circ G \circ L_2 \circ \phi \circ F \circ L_1.$$

We showed that the same public key can be expressed as a composition of only one division map (ϕ) , one multivariate homogeneous polynomial quadratic map $(\hat{G})$, and one multivariate non-homogeneous polynomial quadratic map $(\bar{F})$, as follows:

$$\phi \circ P = \phi \circ \hat{G} \circ \hat{F}.$$

Finally we homogenized the public key, effectively homogenizing the previously non-homogeneous polynomial map $\bar{F}$ into what we call $\hat{F}$.

$$\phi \circ \tilde{P} = \phi \circ \hat{G} \circ \hat{F}$$

Step 2 will follow by applying the partial derivative attack on $\hat{G} \circ \hat{F}$.

5.3 Step 2: Find $\mathcal{F}$

The objective of step 2 is to take the multivariate homogeneous quartic polynomials that form $\tilde{P}$, i.e. $\hat{G} \circ \hat{F}$, and find the vector subspace that we will call $\mathcal{F}_H = \text{Span} \{ \hat{f}_i \circ L_1(x_1, x_2, \dots, x_n) : 1 \leq i \leq n+1 \}$. Now we will treat the polynomials as elements of the polynomial ring $k[x_1, x_2, \dots, x_{n+1}]$ as well as elements of various vector spaces over k . We will attempt to treat these algebraic structures carefully.

This section is divided into two subsections. The first of which describes the direct application of the partial derivative attack on $\hat{G} \circ \hat{F}$. It will include an explanation of what happens as this attack fails. The second subsection will explain our enhancement to the attack, which does succeed.

To begin our more formal presentation, let us define sets $\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3$ and $\mathcal{H}_4$ as follows:

$$\mathcal{H}_1 = \text{Span} \{ x_j : 1 \leq j \leq n+1 \},$$

$$\mathcal{H}_2 = \text{Span} \{ x_j x_k : 1 \leq j, k \leq n+1 \},$$

$$\mathcal{H}_3 = \text{Span} \{ x_j x_k x_l : 1 \leq j, k, l \leq n+1 \},$$

$$\mathcal{H}_4 = \text{Span} \{ x_j x_k x_l x_m : 1 \leq j, k, l, m \leq n+1 \}.$$

The primary objective of this step, again, is to find $\mathcal{F}$ defined as

$$\mathcal{F} = \text{Span} \{ f_i \circ L_1(x_1, x_2, \dots, x_n) : 1 \leq i \leq n+1 \}.$$

To find $\mathcal{F}$, we will first find $\mathcal{F}_H$, which we define as

$$\mathcal{F}_H = \text{Span} \{ \tilde{f}_i(x_1, x_2, \dots, x_{n+1}) : 1 \leq i \leq n+1 \}.$$

Once $\mathcal{F}_H$ is found, we will dehomogenize it by setting $x_{n+1} = 1$. That will give us $\mathcal{F}$.

5.3.1 Direct Application of the Partial Derivative Attack on $\hat{G} \circ \hat{F}$

Computationally, the partial derivative attack is rather simple. We take the multivariate homogeneous quartic polynomials from $\tilde{P}$, namely

$$\tilde{p}_1(x_1, x_2, \dots, x_{n+1}), \tilde{p}_2(x_1, x_2, \dots, x_{n+1}), \dots, \tilde{p}_{n+1}(x_1, x_2, \dots, x_{n+1})$$

and form sets D and R as follows:

$$D = \text{Span} \left\{ \frac{\partial}{\partial x_j} \tilde{p}_i(x_1, x_2, \dots, x_{n+1}) : 1 \leq i, j \leq n+1 \right\},$$

$$R = \{ \theta : \forall 1 \leq i \leq n+1, x_i \theta \in D \}.$$

The original cryptanalysts of 2R claimed that in most cases, $R = \mathcal{F}_H$. Through experiments of our own, we confirmed this claim for cases when the multivariate homogeneous quadratic polynomials

$$\hat{g}_1(x_1, x_2, \dots, x_{n+1}), \hat{g}_2(x_1, x_2, \dots, x_{n+1}), \dots, \hat{g}_{n+1}(x_1, x_2, \dots, x_{n+1})$$

and

$$\hat{f}_1(x_1, x_2, \dots, x_{n+1}), \hat{f}_2(x_1, x_2, \dots, x_{n+1}), \dots, \hat{f}_{n+1}(x_1, x_2, \dots, x_{n+1})$$

were each chosen randomly, meaning that the coefficient of each term was independently chosen from a uniform distribution across the elements of k .

However, in the case of RMPKC, the attack as stated fails. Specifically, $R \neq \mathcal{F}_H$. We do not have a proof as to why it fails, but we do have an enhanced method that works.

To understand the relationship between R and $\mathcal{F}_H$ better, we include the set Λ , defined as follows:

$$\Lambda = \text{Span} \left\{ x_j \hat{f}_i(x_1, x_2, \dots, x_{n+1}) : 1 \leq i, j \leq n+1 \right\}.$$

We have three claims related to the sets $D, R, \mathcal{F}_H$, and Λ . Before we address these claims, let's introduce formally the partial derivative function, which we borrow from calculus: $\frac{\partial}{\partial x_j} : k[x_1, x_2, \dots, x_{n+1}] \longrightarrow k[x_1, x_2, \dots, x_{n+1}]$ as it functions over the real numbers. In the monomial case, consider

$$z \prod_{r=1}^{n+1} x_r^{e_r} \in k[x_1, x_2, \dots, x_{n+1}],$$

$$\frac{\partial}{\partial x_j} \left(z \prod_{r=1}^{n+1} x_r^{e_r} \right) = z e_j x_j^{e_j-1} \prod_{r=1, r \neq j}^{n+1} x_r^{e_r}.$$

Note that $z e_j$ is scalar multiplication with e_j being a non-negative integer. We will not prove it here but the partial derivative function in our case maintains the following properties

as known from calculus:

$$\begin{aligned} & \frac{\partial}{\partial x_j} (z_1 f_1(x_1, x_2, \dots, x_{n+1}) + z_2 f_2(x_1, x_2, \dots, x_{n+1})) \\ &= z_1 \frac{\partial}{\partial x_j} (f_1(x_1, x_2, \dots, x_{n+1})) + z_2 \frac{\partial}{\partial x_j} (f_2(x_1, x_2, \dots, x_{n+1})), \end{aligned}$$

$$\begin{aligned} & \frac{\partial}{\partial x_j} (f_1(x_1, x_2, \dots, x_{n+1}) f_2(x_1, x_2, \dots, x_{n+1})) \\ &= f_1(x_1, x_2, \dots, x_{n+1}) \frac{\partial}{\partial x_j} (f_2(x_1, x_2, \dots, x_{n+1})) \\ &+ f_2(x_1, x_2, \dots, x_{n+1}) \frac{\partial}{\partial x_j} (f_1(x_1, x_2, \dots, x_{n+1})), \end{aligned}$$

and

$$\frac{\partial}{\partial x_j} g_1 \circ F(x_1, x_2, \dots, x_{n+1}) = \sum_{r=1}^{n+1} \left(\frac{\partial}{\partial w_r} g_1(w_1, w_2, \dots, w_{n+1}) \right) \left(\frac{\partial}{\partial x_j} f_r(x_1, x_2, \dots, x_{n+1}) \right)$$

where $F = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{n+1} \end{pmatrix}$ per our notational convention and for $1 \leq x \leq n+1$, w_s is the dummy variable to define g_1 and is evaluated at $w_s = f_s(x_1, x_2, \dots, x_{n+1})$ in the final result.

Here is our first of the set of claims. We call it lemma 1.

LEMMA 1: $D \subseteq \Lambda$

$$D = \text{Span} \left\{ \frac{\partial}{\partial x_j} \tilde{p}_i(x_1, x_2, \dots, x_{n+1}) : 1 \leq i, j \leq n+1 \right\},$$

$$\Lambda = \text{Span} \left\{ x_j \hat{f}_i(x_1, x_2, \dots, x_{n+1}) : 1 \leq i, j \leq n+1 \right\}.$$

To prove $D \subseteq \Lambda$ we will show that for any i, j such that $1 \leq i \leq n+1$ and $1 \leq j \leq n+1$,

$$\frac{\partial}{\partial x_j} \tilde{p}_i(x_1, x_2, \dots, x_{n+1}) \in \Lambda.$$

Proof of Lemma 1:

Take any i, j such that $1 \leq i \leq n+1$ and $1 \leq j \leq n+1$ and consider

$$\begin{aligned} \frac{\partial}{\partial x_j} \tilde{p}_i(x_1, x_2, \dots, x_{n+1}) &= \frac{\partial}{\partial x_j} \hat{g}_i \circ \hat{F}(x_1, x_2, \dots, x_{n+1}) \\ &= \sum_{r=1}^{n+1} \left(\frac{\partial}{\partial w_r} \hat{g}_i(w_1, \dots, w_{n+1}) \right) \left(\frac{\partial}{\partial x_j} \hat{f}_r(x_1, \dots, x_{n+1}) \right) \end{aligned}$$

per the property of the partial derivative function mentioned above. Since $\hat{g}_i$ is homogeneous, $\exists \{ \hat{g}_{i[s,t]} : 1 \leq s \leq t \leq n+1 \}$ such that

$$\hat{g}_i(w_1, w_2, \dots, w_{n+1}) = \sum_{s=1}^{n+1} \sum_{t=s}^{n+1} \hat{g}_{i[s,t]} w_s w_t.$$

and

$$\begin{aligned} \forall 1 \leq r \leq n+1, \hat{g}_i(w_1, w_2, \dots, w_{n+1}) &= \hat{g}_{i[r,r]} w_r w_r + \sum_{s=1}^{r-1} \hat{g}_{i[s,r]} w_s w_r + \sum_{t=r+1}^{n+1} \hat{g}_{i[r,t]} w_r w_t \\ &+ \sum_{s=1, s \neq r}^{n+1} \sum_{t=s, t \neq r}^{n+1} \hat{g}_{i[s,t]} w_s w_t. \end{aligned}$$

Thus

$$\frac{\partial}{\partial w_r} \hat{g}_i(w_1, w_2, \dots, w_{n+1}) = \hat{g}_{i[r,r]} w_r + \hat{g}_{i[r,r]} w_r + \sum_{s=1}^{r-1} \hat{g}_{i[s,r]} w_s + \sum_{t=r+1}^{n+1} \hat{g}_{i[r,t]} w_t,$$

and therefore,

$$\frac{\partial}{\partial w_r} \hat{g}_i(w_1, w_2, \dots, w_{n+1}) \in \text{Span} \{ w_1, w_2, \dots, w_{n+1} \}.$$

Recall that in this case of composition of functions, for each relevant s ,

$$w_s = f_s(x_1, x_2, \dots, x_{n+1})$$

so

$$\begin{aligned} &\left. \frac{\partial}{\partial w_r} \hat{g}_i(w_1, w_2, \dots, w_{n+1}) \right|_{\forall 1 \leq s \leq n+1, w_s = f_s(x_1, x_2, \dots, x_{n+1})} \\ &\in \text{Span} \{ f_1(x_1, x_2, \dots, x_{n+1}), f_2(x_1, x_2, \dots, x_{n+1}), \dots, f_{n+1}(x_1, x_2, \dots, x_{n+1}) \}. \end{aligned}$$

The exact same explanation above can be applied to show that $\forall 1 \leq i, r \leq n+1$,

$$\frac{\partial}{\partial x_r} \hat{f}_i(x_1, x_2, \dots, x_{n+1}) \in \text{Span} \{ x_1, x_2, \dots, x_{n+1} \}.$$

These last two results lead us to

$$\begin{aligned} & \sum_{r=1}^{n+1} \left(\frac{\partial}{\partial w_r} \hat{g}_i(w_1, \dots, w_{n+1}) \right) \left(\frac{\partial}{\partial x_j} \hat{f}_r(x_1, \dots, x_{n+1}) \right) \\ & \in \text{Span} \{ x_s \hat{f}_t(x_1, x_2, \dots, x_{n+1}) : 1 \leq s, t \leq n+1 \} = \Lambda. \end{aligned}$$

Since $\forall 1 \leq i, j \leq n+1$,

$$\frac{\partial}{\partial x_j} \tilde{p}_i(x_1, x_2, \dots, x_{n+1}) \in \Lambda,$$

we have $D \subseteq \Lambda$, concluding our proof.

Our second claim is that $\Lambda \subseteq D \implies \mathcal{F}_H \subseteq R$. We call it Lemma 2.

LEMMA 2: $\Lambda \subseteq D \implies \mathcal{F}_H \subseteq R$

$$\begin{aligned} \Lambda &= \text{Span} \{ x_j \hat{f}_i(x_1, x_2, \dots, x_{n+1}) : 1 \leq i, j \leq n+1 \}. \\ D &= \text{Span} \left\{ \frac{\partial}{\partial x_j} \tilde{p}_i(x_1, x_2, \dots, x_{n+1}) : 1 \leq i, j \leq n+1 \right\}, \\ \mathcal{F}_H &= \text{Span} \{ \tilde{f}_i(x_1, x_2, \dots, x_{n+1}) : 1 \leq i \leq n+1 \}. \\ R &= \{ \theta : \forall 1 \leq i \leq n+1, x_i \theta \in D \}. \end{aligned}$$

Proof of Lemma 2:

Take any $f \in \mathcal{F}_H$. Since $\mathcal{F}_H = \text{Span} \{ \hat{f}_i(x_1, x_2, \dots, x_{n+1}) : 1 \leq i \leq n+1 \}$, $\exists \{ z_i : 1 \leq i \leq n+1 \}$ such that

$$f = \sum_{i=1}^{n+1} z_i \hat{f}_i(x_1, x_2, \dots, x_{n+1}).$$

Now consider $\forall 1 \leq j \leq n+1, x_j f$.

$$\forall 1 \leq j \leq n+1, x_j f = \sum_{i=1}^{n+1} x_j z_i \hat{f}_i(x_1, x_2, \dots, x_{n+1})$$

Since $\Lambda = \text{Span} \left\{ x_j \hat{f}_i(x_1, x_2, \dots, x_{n+1}) : 1 \leq i, j \leq n+1 \right\}$, we get

$$\forall 1 \leq j \leq n+1, x_j f \in \Lambda.$$

Since a condition of the proof is $\Lambda \subseteq D$,

$$\forall 1 \leq j \leq n+1, x_j f \in D.$$

And lastly because $R = \{ \theta : \forall 1 \leq i \leq n+1, x_i \theta \in D \}$, by definition, $f \in R$.

So our proof that $\Lambda \subseteq D \implies \mathcal{F}_H \subseteq R$ is complete.

Our third and last claim here is $\Lambda = D, R \subseteq \mathcal{F}_H \implies \mathcal{F}_H = R$. We call it Lemma 3 but the result follows directly from Lemma 1 ($D \subseteq \Lambda$) and Lemma 2 ($\Lambda \subseteq D \implies \mathcal{F}_H \subseteq R$).

LEMMA 3: $\Lambda = D, R \subseteq \mathcal{F}_H \implies \mathcal{F}_H = R$

PROOF of Lemma 3: omitted; this lemma follows directly from previous results.

This last claim is also the key to the success of the partial derivative attack. If $\mathcal{F}_H = R$ we succeed; otherwise, we fail.

In our cryptanalysis, we calculate D and R . Early experiments repeatedly (100% of tests run) revealed that we get $R \subseteq \mathcal{F}_H$ but not $\Lambda \subseteq D$. Again we have no proof as to why we don't get $\Lambda \subseteq D$, but we know that it has something to do with the structure of the polynomials in RMPKC. The experiments ran using polynomials chosen completely randomly (each coefficient chosen randomly from k) confirmed the success found by the cryptanalysts of 2R, with $\Lambda \subseteq D$. And when the polynomials are formed according to RMPKC, we do not get $\Lambda \subseteq D$.

5.3.2 Our Enhancement to the Partial Derivative Attack

It is the structure of the multivariate quadratic polynomials of the maps G and F that lead to the problems described above. We know this because the partial derivative attack works with a success rate of nearly 1 when the quadratics are chosen randomly, that is the

coefficient of each term in the polynomial is selected randomly from a uniform distribution across the elements of K .

Furthermore, this structure mentioned above is somewhat hidden by the linear transformations L_1, L_2 , and L_3 . Further mixing the variables as an attempt to overcome the problem does not work. A more sophisticated approach is required.

The key to our approach is to project the entire cryptosystem onto subspaces according to $x_s = 0$. We do this for each $1 \leq s \leq n + 1$. In each case, we define the following:

$$D_s = \text{Span} \left\{ \frac{\partial}{\partial x_j} \tilde{p}_i(x_1, \dots, x_{s-1}, 0, x_{s+1}, \dots, x_{n+1}) : 1 \leq i, j \leq n + 1 \right\}$$

$$R_s = \{ \theta \in \mathcal{H}_2 : \forall 1 \leq i \leq n + 1 (i \neq s), x_i \theta \in D_s \} .$$

$$\mathcal{F}_s = \text{Span} \{ \hat{f}(x_1, \dots, x_{s-1}, 0, x_{s+1}, \dots, x_{n+1}) : \forall \hat{f} \in \mathcal{F}_H \} ,$$

and

$$\Lambda_s = \{ x_i \hat{f} : 1 \leq i \leq n + 1 (i \neq s), \hat{f} \in \mathcal{F}_s \} .$$

We have chosen the computationally easiest subspaces using $x_s = 0$. Other subplanes could be used, such as in the form $\sum_{j=1}^{n+1} z_j x_j = z_0$, for some $z_0, z_1, \dots, z_{n+1} \in k$. Of course we would have to generate several of these and consider linear independence among them. But since RMPKC randomly chooses elements for k as the coefficients of L_1 , the variables $x_1, x_2, \dots, x_n$ have already been mixed up so our choice of $x_s = 0$ for the projections is just as good as a more complex one.

The three lemmas proven previously, namely

1. $D \subseteq \Lambda$,
2. $\Lambda \subseteq D \implies \mathcal{F}_H \subseteq R$, and
3. $R \subseteq \mathcal{F}_H, \Lambda = D \implies \mathcal{F}_H = R$,

apply directly to give us

1. $D_s \subseteq \Lambda_s$,
2. $\Lambda_s \subseteq D_s \implies \mathcal{F}_s \subseteq R_s$, and
3. $R_s \subseteq \mathcal{F}_s, \Lambda_s = D_s \implies \mathcal{F}_s = R_s$.

The proof would be completely redundant with very messy notation. The doubtful reader might find comfort in thinking that the projected cryptosystem is merely a reduction in variables from $n + 1$ to n . Any changes to the specific structures of the polynomials of RMPKC made by this projection have no absolutely impact on the proofs.

To continue our enhancement, we need to combine the results from the projections. At this point we have for each $1 \leq s \leq n + 1$, R_s .

Our next step is to define R_s^+ .

$$\forall 1 \leq s \leq n + 1, R_s^+ = R_s + \text{Span} \{ x_s x_i : 1 \leq i \leq n + 1 \}$$

And finally we define R^+ .

$$R^+ = \bigcap_{s=1}^{n+1} R_s^+$$

Unfortunately, we are not able to explain why this projection works to make $R^+ = \mathcal{F}_H$ when $R \neq \mathcal{F}_H$. As a matter of fact, $R^+ = \mathcal{F}_H$ works out only for $n = 5$ and $n = 6$. For greater values of n , we need to generalize the method.

We can prove one thing: that $R \subseteq R^+$. This fact suggests that we are not going in the wrong direction, but does not explain why we are going in the right direction.

LEMMA 4: $R \subseteq R^+$

Proof of Lemma 4

Take any $f \in R$. Take any s in $\{ 1, 2, \dots, n + 1 \}$. We can express f as the following:

$$f(x_1, x_2, \dots, x_{n+1}) = \sum_{i=1}^{n+1} \sum_{j=i}^{n+1} f_{[i,j]} x_i x_j.$$

Furthermore,

$$f \in R \implies \forall 1 \leq r \leq n+1, x_r f \in D.$$

Now take any $1 \leq r \leq n+1$ such that $r \neq s$.

$$x_r \sum_{i=1}^{n+1} \sum_{j=i}^{n+1} f_{[i,j]} x_i x_j \in D$$

When we consider the projection of $x_r f$ onto $x_s = 0$, we get

$$x_r \sum_{\substack{i=1 \\ i \neq s}}^{n+1} \sum_{\substack{j=i \\ j \neq s}}^{n+1} f_{[i,j]} x_i x_j.$$

By construction of D_s , we may also define it by $D_s = \{ \theta(x_1, \dots, x_{s-1}, 0, x_{s+1}, \dots, x_{n+1}) : \theta(x_1, \dots, x_{s-1}, x_s, x_{s+1}, \dots, x_{n+1}) \in D \}$. Elements of D_s are made by taking elements from D and setting $x_s = 0$.

Now it is straightforward to see that $x_r f(x_1, \dots, x_{s-1}, 0, x_{s+1}, \dots, x_{n+1}) \in D_s$.

Now since we have shown for any given $r \neq s$,

$$x_r f(x_1, \dots, x_{s-1}, 0, x_{s+1}, \dots, x_{n+1}) \in D_s,$$

$f \in R_s$ is true by definition of R_s . Thus we get the result $R \subseteq R_s$.

And now that we have proven that for any given s , $R \subseteq R_s$, we get $R \subseteq R^+$. Our proof is concluded.

Again this fact that $R \subseteq R^+$ does not explain why our enhancement succeeds. The only evidence we have is experimental. The argument that we give is that the structure of the polynomials in RMPKC prevent the partial derivative attack from working directly. Projecting the cryptosystem to $x_s = 0$ breaks up this structure somehow.

Our enhanced method to find $\mathcal{F}_H$ works consistently for $n = 5$ and $n = 6$. However for $7 \leq n \leq 14$ it fails. We have a generalization of the enhancement that works for these higher values of n .

For $n = 7$, we use $x_{s_1} = 0 = x_{s_2}$ for the projections, for each $1 \leq s_1, s_2 \leq n + 1$ with $s_1 \neq s_2$. Following the same manner we form D_{s_1, s_2} and R_{s_1, s_2} . Then we let $R_{s_1, s_2}^+ = R_{s_1, s_2} + \text{Span} \{ x_{s_1} x_i : 1 \leq i \leq n + 1 \} + \text{Span} \{ x_{s_2} x_i : 1 \leq i \leq n + 1 \}$. We finish with $R^{++} = \bigcap_{\substack{1 \leq s_1, s_2 \leq n+1 \\ s_1 \neq s_2}} R_{s_1, s_2}^+$.

With consistency, we do get $R^{++} = \mathcal{F}_H$.

For $n \geq 8$, we use $x_{s_1} = 0 = x_{s_2} = 0 = x_{s_3}$ for each $1 \leq s_1, s_2, s_3 \leq n + 1$, where $s_1 \neq s_2 \neq s_3 \neq s_1$. Following the same pattern we form D_{s_1, s_2, s_3} and R_{s_1, s_2, s_3} . Then we let $R_{s_1, s_2, s_3}^+ = R_{s_1, s_2, s_3} + \text{Span} \{ x_{s_1} x_i : 1 \leq i \leq n + 1 \} + \text{Span} \{ x_{s_2} x_i : 1 \leq i \leq n + 1 \} + \text{Span} \{ x_{s_3} x_i : 1 \leq i \leq n + 1 \}$. Finally we let $R^{+++} = \bigcap_{\substack{1 \leq s_1, s_2, s_3 \leq n+1 \\ s_1 \neq s_2 \neq s_3 \neq s_1}} R_{s_1, s_2, s_3}^+$.

Once again through our results we consistently get $\mathcal{F} = R^{+++}$.

We have shown the pattern that enables us to find $\mathcal{F}_H$. In the application of the attack, we start by calculating R^+ and see if it is of the right dimension $(n + 1)$. If that doesn't work, we calculate R^{++} . If that still does not work, we continue on with R^{+++} . In our experiments, we saw a clear relationship between n and how far we needed to go through these approaches; this relationship is as we've stated.

We ran tests which consistently succeeded with several values of n up to 14. In 100% of the experiments, we found $\mathcal{F}_H$. The final task for step 2 is to obtain $\mathcal{F}$ from $\mathcal{F}_H$. We set $x_{n+1} = 1$ into each polynomial of $\mathcal{F}_H$. Formally,

$$\mathcal{F} = \{ f(x_1, \dots, x_n, 1) : \forall f(x_1, \dots, x_n, x_{n+1}) \in \mathcal{F}_H \}.$$

$$\mathcal{F} = \text{Span} \{ f_i \circ L_1(x_1, x_2, \dots, x_n) : \forall 1 \leq i \leq n + 1 \}.$$

We have successfully enhanced the partial derivative attack to find $\mathcal{F}$ given the $n + 1$ component polynomials of the public key, P . This concludes step 2.

5.4 Step 3: Find $F' \circ L'_1$

Recall from step one that the formulation of the public key which we are using is the following:

$$\phi \circ P = \phi \circ \tilde{G} \circ \hat{F},$$

where

$$\hat{G} : k^{n+1} \longrightarrow k^{n+1}, \hat{G}(x_1, \dots, x_{n+1}) = \begin{pmatrix} \hat{g}_1(x_1, \dots, x_{n+1}) \\ \hat{g}_2(x_1, \dots, x_{n+1}) \\ \vdots \\ \hat{g}_{n+1}(x_1, \dots, x_{n+1}) \end{pmatrix},$$

where $\forall 1 \leq i \leq n+1$, $\hat{g}_i(x_1, \dots, x_n)$ is a multivariate homogenous quadratic polynomial, and

$$\bar{F} : k^n \longrightarrow k^{n+1}, \bar{F}(x_1, \dots, x_n) = \begin{pmatrix} \bar{f}_1(x_1, \dots, x_n) \\ \bar{f}_2(x_1, \dots, x_n) \\ \vdots \\ \bar{f}_{n+1}(x_1, \dots, x_n) \end{pmatrix}$$

where $\forall 1 \leq i \leq n+1$, $\bar{f}_i(x_1, \dots, x_n)$ is a multivariate quadratic polynomial (not necessarily homogeneous).

In step two, we found $\mathcal{F}$, which is defined by the following:

$$\begin{aligned} \mathcal{F} &= \text{Span} \{ f_i \circ L_1(x_1, x_2, \dots, x_n) : 1 \leq i \leq n+1 \} \\ &= \text{Span} \{ \bar{f}_i(x_1, x_2, \dots, x_n) : 1 \leq i \leq n+1 \}. \end{aligned}$$

Now in step three, we shall find a linear transformation L'_1 and a multivariate quadratic polynomial map F' as follows:

$$L'_1 : k^n \longrightarrow k^n, L'_1(x_1, \dots, x_n) = \begin{pmatrix} L'_{1,1}(x_1, \dots, x_n) \\ L'_{1,2}(x_1, \dots, x_n) \\ \vdots \\ L'_{1,n}(x_1, \dots, x_n) \end{pmatrix}$$

$$F' : k^n \longrightarrow k^{n+1}, F'(x_1, \dots, x_n) = \begin{pmatrix} f'_1(x_1, \dots, x_n) \\ f'_2(x_1, \dots, x_n) \\ \vdots \\ f'_{n+1}(x_1, \dots, x_n) \end{pmatrix}$$

where $\forall 1 \leq i \leq n+1$, $L'_1(x_1, \dots, x_n)$ is a degree one polynomial and $f'_i(x_1, \dots, x_n)$ is a multivariate quadratic polynomial. Furthermore,

$$Span \{ f'_i \circ L'_1(x_1, x_2, \dots, x_n) : 1 \leq i \leq n+1 \} = \mathcal{F}$$

and

$$f_{n+1} \circ L_1(x_1, x_2, \dots, x_n) \in Span \{ f'_{n+1} \circ L'_1(x_1, x_2, \dots, x_n) \}.$$

Additionally, the multivariate quadratic polynomials $f'_i(x_1, x_2, \dots, x_n)$, $\forall 1 \leq i \leq n+1$ form the map F' which legitimately fits into the RMPKC structure in the same manner as the multivariate quadratic polynomial map F , which was used to create the system. Furthermore, we construct these multivariate quadratic polynomials $f'_i(x_1, x_2, \dots, x_n)$ in such a way so that we can use the RMPKC inversion process on them. The linear transformation L'_1 , the multivariate quadratic polynomial map F' , and the means to invert F' become part of the alternate private key, which is the end goal of this cryptanalysis.

The process to construct F' and L'_1 begins with the core transformation. We will first define

$$L'_{1,n-1}, L'_{1,n}, F'_{n-1}, F'_n, F'_{n+1}.$$

5.4.1 Find the Core Transformation of F'

Afterward, we shall find the other components of L'_1 and F' that relate to the sequential part of F , but for now let us concentrate on the core transformation.

Consider the polynomials of the core transformation, which are constructed with only two variables, x_{n-1} and x_n such that

$$f_{n-1}(x_{n-1}, x_n), f_n(x_{n-1}, x_n), f_{n+1}(x_{n-1}, x_n) \in \text{Span} \{ x_{n-1}x_n, x_{n-1}, x_n, 1 \}.$$

The space formed by the span of the three polynomials has a two-dimensional subspace contained in $\text{Span} \{ x_{n-1}, x_n, 1 \}$. This is true because the three polynomials under consideration are linearly independent. The key here is that the polynomials of this subspace are of dimension one, i.e. linear.

Note that there is a nonzero probability that the two-dimensional subspace in question includes 1. In this case our attack as implemented fails. A backup method in this case has not been developed. However, due to the construction of the polynomials in question, the probability of this occurrence is $\frac{1}{|K|}$, which again, is very small.

Now consider $\mathcal{F} = \text{Span} \{ f_i \circ L_1(x_1, x_2, \dots, x_n) : 1 \leq i \leq n+1 \}$. Since L_1 is a linear transformation, $\mathcal{F}$ must also have a two-dimensional subspace consisting only of polynomials of dimension one. Note that the polynomials in this subspace are not restricted to the variables x_{n-1} and x_n . The composition of F with L_1 effectively mixes up the variables so that the resulting polynomials related to the core transformation should be expressed as $f_{n-1} \circ L_1(x_1, \dots, x_n)$, $f_n \circ L_1(x_1, \dots, x_n)$, and $f_{n+1} \circ L_1(x_1, \dots, x_n)$.

Nevertheless, Gaussian elimination easily locates the two-dimensional subspace of $\mathcal{F}$ con-

sisting of polynomials of degree one. Let $\hat{L}_{1,n-1}$ and $\hat{L}_{1,n}$ be two homogeneous linear polynomials such that this two-dimensional subspace of $\mathcal{F}$ is contained in $Span \{ \hat{L}_{1,n-1}, \hat{L}_{1,n}, 1 \}$. Essentially, we are finding a basis for this subspace while disregarding (temporarily) the constant terms.

One can see that $Span \{ \hat{L}_{1,n-1}, \hat{L}_{1,n} \} = Span \{ L_{1,n-1}, L_{1,n} \}$.

Recall that

$$f_{n-1}(x_{n-1}, x_n), f_n(x_{n-1}, x_n), f_{n+1}(x_{n-1}, x_n) \in Span \{ x_{n-1}x_n, x_{n-1}, x_n, 1 \}.$$

This fact implies that

$$Span \{ f_{n-1} \circ L_1, f_n \circ L_1, f_{n+1} \circ L_1 \} \subset Span \{ L_{1,n-1}L_{1,n}, L_{1,n-1}, L_{1,n}, 1 \}.$$

where each element listed is a polynomial over $(x_1, x_2, \dots, x_n)$.

We do not have $L_{1,n-1}$ and $L_{1,n}$ in our cryptanalysis, but we do have $\hat{L}_{1,n-1}$ and $\hat{L}_{1,n}$. And since $Span \{ \hat{L}_{1,n-1}, \hat{L}_{1,n} \} = Span \{ L_{1,n-1}, L_{1,n} \}$, we get

$$Span \{ f_{n-1} \circ L_1, f_n \circ L_1, f_{n+1} \circ L_1 \} \subset Span \{ \hat{L}_{1,n-1}\hat{L}_{1,n}, \hat{L}_{1,n-1}^2, \hat{L}_{1,n}^2, \hat{L}_{1,n-1}, \hat{L}_{1,n}, 1 \}$$

where each element listed is again a polynomial over $(x_1, x_2, \dots, x_n)$.

We want to find $L'_{1,n-1}$ and $L'_{1,n}$ where $L'_{1,n-1} = \hat{L}_{1,n-1} + s\hat{L}_{1,n}$ and $L'_{1,n} = \hat{L}_{1,n-1} + t\hat{L}_{1,n}$ such that

$$Span \{ f_{n-1} \circ L_1, f_n \circ L_1, f_{n+1} \circ L_1 \} \subset Span \{ L'_{1,n-1}L'_{1,n}, L'_{1,n-1}, L'_{1,n}, 1 \}$$

where each element listed is again a polynomial over $(x_1, x_2, \dots, x_n)$.

Fortunately, we can calculate

$$Span \{ \hat{L}_{1,n-1}\hat{L}_{1,n}, \hat{L}_{1,n-1}^2, \hat{L}_{1,n}^2, \hat{L}_{1,n-1}, \hat{L}_{1,n}, 1 \} \cap \mathcal{F}$$

which is equal to

$$Span \{ f_{n-1} \circ L_1, f_n \circ L_1, f_{n+1} \circ L_1 \}$$

where each of the elements listed is a polynomial over $x_1, \dots, x_n$.

Furthermore, since we have $\hat{L}_{1,n-1}$ and $\hat{L}_{1,n}$, we can identify a third element in $Span \{ f_{n-1} \circ L_1, f_n \circ L_1, f_{n+1} \circ L_1 \}$ which is in $Span \{ \hat{L}_{1,n-1}^2, \hat{L}_{1,n}^2, \hat{L}_{1,n-1}\hat{L}_{1,n}, 1 \}$. Let's pick such a third element θ in such a way that it can be expressed as $\theta = \hat{L}_{1,n-1}^2 + a\hat{L}_{1,n-1}\hat{L}_{1,n} + b\hat{L}_{1,n}^2 + c$.

We need to get $L'_{1,n-1}L'_{1,n} \in Span \{ \theta, 1 \}$ to enable

$$Span \{ f_{n-1} \circ L_1, f_n \circ L_1, f_{n+1} \circ L_1 \} \subset Span \{ L'_{1,n-1}L'_{1,n}, L'_{1,n-1}, L'_{1,n}, 1 \}.$$

To get $L'_{1,n-1}L'_{1,n} \in Span \{ \theta, 1 \}$ we need

$$(\hat{L}_{1,n-1} + s\hat{L}_{1,n})(\hat{L}_{1,n-1} + t\hat{L}_{1,n}) = \hat{L}_{1,n-1}^2 + a\hat{L}_{1,n-1}\hat{L}_{1,n} + b\hat{L}_{1,n}^2.$$

So we need

$$\hat{L}_{1,n-1}^2 + (s+t)\hat{L}_{1,n-1}\hat{L}_{1,n} + st\hat{L}_{1,n}^2 = \hat{L}_{1,n-1}^2 + a\hat{L}_{1,n-1}\hat{L}_{1,n} + b\hat{L}_{1,n}^2,$$

which implies that we need $s+t = a$ and $st = b$. By substitution of $t = a - s$ we get $s(a - s) = b$, or $s^2 - as + b = 0$. We know a and b and we want to find s . In a finite field of char 2, the solution of this equation, if it exists, is not difficult. We know that the solution exists because the public key of the cryptosystem was formed in exactly this way. In our case, there are two solutions. One solution leads to $L'_{1,n-1} \in Span \{ L_{1,n-1} \}$ and $L'_{1,n} \in Span \{ L_{1,n} \}$. The other solution leads to $L'_{1,n-1} \in Span \{ L_{1,n} \}$ and $L'_{1,n} \in Span \{ L_{1,n-1} \}$. In the cryptoanalysis, we never know which solution we get because we don't have access to L_1 ; but it doesn't matter to us. We will be able to fully construct an alternate private key using the solution we find.

After finding s , we calculate $t = a - s$ and form $L'_{1,n-1} = \hat{L}_{1,n-1} + s\hat{L}_{1,n}$ and $L'_{1,n} = \hat{L}_{1,n-1} + t\hat{L}_{1,n}$. With $L'_{1,n-1}$ and $L'_{1,n}$ it is easy to find f'_{n-1}, f'_n , and f'_{n+1} such that

$$Span \{ f'_{n-1}(L'_{1,n-1}, L'_{1,n}), f'_n(L'_{1,n-1}, L'_{1,n}), f'_{n+1}(L'_{1,n-1}, L'_{1,n}) \}$$

$$= \text{Span} \{ f_{n-1}(L_{1,n-1}, L_{1,n}), f_n(L_{1,n-1}, L_{1,n}), f_{n+1}(L_{1,n-1}, L_{1,n}) \} .$$

Essentially we could use any basis elements of

$$\text{Span} \{ f_{n-1}(L_{1,n-1}, L_{1,n}), f_n(L_{1,n-1}, L_{1,n}), f_{n+1}(L_{1,n-1}, L_{1,n}) \}$$

and solve for the coefficients for each term in the quadratic polynomials of each of f'_{n-1} , f'_n , and f'_{n+1} , but recall that we also need $f'_{n+1} \in \text{Span} \{ f_{n+1} \}$ because of the special requirement associated with the division employed in this cryptosystem.

The trick to find $\text{Span} \{ f_{n+1} \}$ is to recognize that $f_{n+1} \circ L_1$ divides a one dimensional subspace of $\mathcal{P} = \text{Span} \{ p_1, \dots, p_{n+1} \}$, where p_1 through p_{n+1} are the quartic polynomials of the public key. Here we explain why this is true.

Recall the construction of the public key:

$$\phi \circ P = L_3 \circ \phi \circ G \circ L_2 \circ \phi \circ F \circ L_1.$$

Let $\mathcal{G} = \text{Span} \{ g_1, \dots, g_{n+1} \}$ where $g_1, \dots, g_{n+1}$ are the quadratic polynomial components of G . $\mathcal{G}$, like $\mathcal{F}$, has a two-dimensional linear subspace. Therefore, $\mathcal{G}$ has a one-dimensional homogeneous linear subspace. Let g^* be a basis element of this homogeneous linear subspace.

As described in step one of our cryptanalysis, let $\tilde{G}$ be the map defined as follows:

$$\tilde{G} : k^{n+1} \longrightarrow k^{n+1}, \tilde{G}(x_1, \dots, x_{n+1}) = \begin{pmatrix} \tilde{g}_1(x_1, \dots, x_{n+1}) \\ \vdots \\ \tilde{g}_{n+1}(x_1, \dots, x_{n+1}) \end{pmatrix}$$

where $\forall 1 \leq i \leq n+1, \tilde{g}_i(x_1, \dots, x_{n+1}) = x_{n+1}^2 g_i(\frac{x_1}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}})$ and $\forall 1 \leq i \leq n+1, g_i$ is the component quadratic polynomial of the map G .

Now let's define $\tilde{g}^*(x_1, \dots, x_{n+1}) = x_{n+1}^2 g^*(\frac{x_1}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}})$. Since g^* was a basis element of a one-dimensional homogeneous linear subspace of $\mathcal{G}$, $\tilde{g}^*$ is a basis element of a one-dimensional homogeneous quadratic subspace of $\text{Span} \{ \tilde{g}_1, \dots, \tilde{g}_{n+1} \}$. By construction, $x_{n+1} \mid \tilde{g}^*(x_1, \dots, x_{n+1})$.

Consider $\tilde{g}^* \circ \tilde{L}_2 \circ F \circ L_1$ where two formulations of the public key as described in step one of the cryptanalysis are

$$\phi \circ P = L_3 \circ \phi \circ G \circ \phi \circ \tilde{L}_2 \circ F \circ L_1$$

and

$$\phi \circ P = L_3 \circ \phi \circ \tilde{G} \circ \tilde{L}_2 \circ F \circ L_1.$$

Since the $(n+1)^{st}$ component of the domain of $\tilde{L}_2$ is invariant under $\tilde{L}_2$, we can state

$$\tilde{g}^*(\tilde{L}_2 \circ F \circ L_1(x_1, \dots, x_n)) = [f_{n+1} \circ L_1(x_1, \dots, x_n)] [g^* \circ L_2 \circ F \circ L_1(x_1, \dots, x_n)].$$

Therefore $\mathcal{P}$ has a subspace divided by $f_{n+1} \circ L_1$, which is an element of $\mathcal{F}$.

Note that we have not proven uniqueness of a subspace of $\mathcal{P}$ which is divisible by elements of $Span \{ L'_{1,n-1}, L'_{1,n} \}$. Our experiments find the exact subspace every time, but as we will see in the small-scale example, there is a possibility that more than one such subspace exists. We conclude that again the randomness involved in the construction of RMPKC renders this possibility very small.

We find this linear subspace of $\mathcal{P}$ and identify f'_{n+1} exactly using $L'_{1,n-1}$ and $L'_{1,n}$ such that $f'_{n+1} \circ L_1^*$ divides the linear subspace of $\mathcal{P}$, where L_1^* represents the $(n-1)^{st}$ and n^{th} components of L'_1 , which are the only components of L'_1 which have already found and which are used in the core transformation.

We can pick any additional polynomials to complete a basis for

$$Span \{ f_{n-1}(L_{1,n-1}, L_{1,n}), f_n(L_{1,n-1}, L_{1,n}), f_{n+1}(L_{1,n-1}, L_{1,n}) \}.$$

As part of the cryptanalysis, we need to find way to invert this core transformation we are creating for our alternate private key. If the basis elements that we choose were to lead to failure in our effort to define the inversion, we would choose other polynomials (in addition

to $f'_{n+1} \circ L_1'^*$) to complete a basis for

$$\text{Span} \{ f_{n-1}(L_{1,n-1}, L_{1,n}), f_n(L_{1,n-1}, L_{1,n}), f_{n+1}(L_{1,n-1}, L_{1,n}) \}.$$

We used a randomized approach to completing the basis, and found that it was successful every time.

5.4.2 Create an Inversion Process for the Core Transformation of $\mathbf{F}$

Recall from chapter 2, section 3, the formation of the core transformation in RMPKC. To construct the core transformation, we randomly choose 12 elements from k , namely $\alpha_1, \dots, \alpha_6$ and $\beta_1, \dots, \beta_6$. We then calculate the coefficients of the core transformation polynomials. These coefficients are named $\tau_{i,j}$ for $n-1 \leq i \leq n+1$ and $1 \leq j \leq 4$ and are calculated by the following:

$$\begin{aligned} \tau_{n-1,1} &= \alpha_6\beta_4 - \alpha_4\beta_6 & \tau_{n,1} &= \alpha_5\beta_6 - \alpha_6\beta_5 & \tau_{n+1,1} &= \alpha_4\beta_5 - \alpha_5\beta_4 \\ \tau_{n-1,2} &= \alpha_1\beta_6 - \alpha_3\beta_4 & \tau_{n,2} &= \alpha_3\beta_5 - \alpha_2\beta_6 & \tau_{n+1,2} &= \alpha_2\beta_4 - \alpha_1\beta_5 \\ \tau_{n-1,3} &= \alpha_4\beta_3 - \alpha_6\beta_1 & \tau_{n,3} &= \alpha_6\beta_2 - \alpha_5\beta_3 & \tau_{n+1,3} &= \alpha_5\beta_1 - \alpha_4\beta_2 \\ \tau_{n-1,4} &= \alpha_3\beta_1 - \alpha_1\beta_3 & \tau_{n,4} &= \alpha_2\beta_3 - \alpha_3\beta_2 & \tau_{n+1,4} &= \alpha_1\beta_2 - \alpha_2\beta_1 \end{aligned}$$

Finally the core transformation polynomials are constructed as follows:

$$\begin{aligned} f_{n-1}(x_{n-1}, x_n) &= \tau_{n-1,1}x_{n-1}x_n + \tau_{n-1,2}x_{n-1} + \tau_{n-1,3}x_n + \tau_{n-1,4} \\ f_n(x_{n-1}, x_n) &= \tau_{n,1}x_{n-1}x_n + \tau_{n,2}x_{n-1} + \tau_{n,3}x_n + \tau_{n,4} \\ f_{n+1}(x_{n-1}, x_n) &= \tau_{n+1,1}x_{n-1}x_n + \tau_{n+1,2}x_{n-1} + \tau_{n+1,3}x_n + \tau_{n+1,4} \end{aligned}$$

In our case, we have found the polynomials, so we have each of the coefficients (the τ 's). We need to now find appropriate values for each of $\alpha'_1, \dots, \alpha'_6$ and $\beta'_1, \dots, \beta'_6$ so that we may invert the core transformation using

$$(x_{n-1}, x_n) = \left(\frac{\alpha'_1 y_n + \alpha'_2 y_{n-1} + \alpha'_3}{\alpha'_4 y_n + \alpha'_5 y_{n-1} + \alpha'_6}, \frac{\beta'_1 y_n + \beta'_2 y_{n-1} + \beta'_3}{\beta'_4 y_n + \beta'_5 y_{n-1} + \beta'_6} \right).$$

We are careful here because the relation between the x 's and y 's in the above equation is given by

$$y_{n-1}(x_{n-1}, x_n) = \frac{f_{n-1}(x_{n-1}, x_n)}{f_{n+1}(x_{n-1}, x_n)}, \quad y_n(x_{n-1}, x_n) = \frac{f_n(x_{n-1}, x_n)}{f_{n+1}(x_{n-1}, x_n)}.$$

By considering separately the components, and by recognizing that we are working with our alternate key, we will employ the symbols associated with the rest of the flow of the cryptanalysis, namely f' , α' and β' . That gives us

$$x_n = \frac{\alpha'_1 f'_n(x_{n-1}, x_n) + \alpha'_2 f'_{n-1}(x_{n-1}, x_n) + \alpha'_3 f'_{n+1}(x_{n-1}, x_n)}{\alpha'_4 f'_{n-1}(x_n, x_n) + \alpha'_5 f'_{n-1}(x_{n-1}, x_n) + \alpha'_6 f'_{n+1}(x_{n-1}, x_n)}$$

and

$$x_{n-1} = \frac{\beta'_1 f'_n(x_{n-1}, x_n) + \beta'_2 f'_{n-1}(x_{n-1}, x_n) + \beta'_3 f'_{n+1}(x_{n-1}, x_n)}{\beta'_4 f'_n(x_{n-1}, x_n) + \beta'_5 f'_{n-1}(x_{n-1}, x_n) + \beta'_6 f'_{n-1}(x_{n+1}, x_n)}.$$

where

$$f'_{n-1}(x_{n-1}, x_n) = \tau'_{n-1,1} x_{n-1} x_n + \tau'_{n-1,2} x_{n-1} + \tau'_{n-1,3} x_n + \tau'_{n-1,4}$$

$$f'_n(x_{n-1}, x_n) = \tau'_{n,1} x_{n-1} x_n + \tau'_{n,2} x_{n-1} + \tau'_{n,3} x_n + \tau'_{n,4}$$

$$f'_{n+1}(x_{n-1}, x_n) = \tau'_{n+1,1} x_{n-1} x_n + \tau'_{n+1,2} x_{n-1} + \tau'_{n+1,3} x_n + \tau'_{n+1,4}$$

and where we have values for each listed $\tau_{i,j}$ from the previous step of the cryptanalysis.

By cross-multiplication we get

$$\begin{aligned} & x_n \left[\alpha'_4 f'_n(x_{n-1}, x_n) + \alpha'_5 f'_{n-1}(x_{n-1}, x_n) + \alpha'_6 f'_{n+1}(x_{n-1}, x_n) \right] \\ &= \alpha'_1 f'_n(x_{n-1}, x_n) + \alpha'_2 f'_{n-1}(x_{n-1}, x_n) + \alpha'_3 f'_{n+1}(x_{n-1}, x_n) = 0 \end{aligned}$$

which can be used to generate six further equations based upon the relevant coefficients of each of $1, x_{n-1}, x_n, (x_{n-1})^2, x_{n-1}x_n$, and $(x_{n-1})^2x_n$ in the above equation. The only unknowns in these six equations are $\alpha'_1, \dots, \alpha'_6$, and these equations are all linear. Therefore these six equations can be solved simultaneously to find $\alpha'_1, \dots, \alpha'_6$.

As you can see from the design of RMPKC, these α 's are not unique. Using any of the available solutions works fine for our cryptanalysis.

We perform the same process using

$$\begin{aligned} & x_{n-1} \left[\beta'_4 f'_n(x_{n-1}, x_n) + \beta'_5 f'_{n-1}(x_{n-1}, x_n) + \beta'_6 f'_{n+1}(x_{n-1}, x_n) \right] \\ &= \beta'_1 f'_n(x_{n-1}, x_n) + \beta'_2 f'_{n-1}(x_{n-1}, x_n) + \beta'_3 f'_{n+1}(x_{n-1}, x_n) = 0 \end{aligned}$$

to find appropriate values for $\beta'_1, \dots, \beta'_6$. And these resultant $\alpha'_1, \dots, \alpha'_6, \beta'_1, \dots, \beta'_6$ successfully fill the role of inverting the core transformation as part of our alternate private key.

5.4.3 Find the Sequential Part of F'

Now that we have found the core transformation of F' , and the related components of the linear transformation L'_1 , we now determine the sequential part of F' and its related components of L'_1 . Recall the structure of the sequential part of F . Each component of F is a multivariate quadratic polynomial divided by f_{n+1} , also a multivariate quadratic polynomial. In our construction of F' , we are using the same structure as F . We have already found our f'_{n+1} , which lies in $Span \{ f_{n+1} \}$.

$$\forall 1 \leq i \leq n-2, f_i(x_i, \dots, x_n) = a_i(x_{i+1}, \dots, x_n)x_i + b_i(x_{i+1}, \dots, x_n).$$

We construct the polynomials of F in the same manner. Specifically, to find f'_j , $2 \leq j \leq n-2$, we look for the intersection of $\mathcal{F}$ and

$$\begin{aligned} & Span \{ L'_j L'_{j+1}, L'_j L'_{j+2}, \dots, L'_j L'_n \} + Span \{ (L'_{j+1})^2, (L'_{j+2})^2, \dots, (L'_n)^2 \} \\ &+ Span \{ L'_{j+1} L'_{j+2}, L'_{j+1} L'_{j+3}, \dots, L'_{j+1} L'_n, L'_{j+2} L'_{j+3}, L'_{j+2} L'_{j+4}, \dots, L'_{j+2} L'_n, \dots, L'_{n-1} L'_n \} \\ &+ Span \{ L'_j, L'_{j+1}, \dots, L'_n, 1 \}. \end{aligned}$$

We find these polynomials starting with $j = n-2$, then $j = n-1$, and continue working our way down to $j = 2$. So at any step in this process, we already have identified $L'_{j+1}, L'_{j+2}, \dots, L'_n$.

We don't know L'_j , so we look at the intersection of $\mathcal{F}$ and the following:

$$\begin{aligned} & \text{Span} \{ x_1 L'_{j+1}, x_2 L'_{j+1}, \dots, x_n L'_{j+1}, x_1 L'_{j+2}, x_2 L'_{j+2}, \dots, x_n L'_{j+2}, \dots, x_1 L'_n, x_2 L'_n, \dots, x_n L'_n \} \\ & + \text{Span} \{ (L'_{j+1})^2, (L'_{j+2})^2, \dots, (L'_n)^2 \} \\ & + \text{Span} \{ L'_{j+1} L'_{j+2}, L'_{j+1} L'_{j+3}, \dots, L'_{j+1} L'_n, L'_{j+2} L'_{j+3}, L'_{j+2} L'_{j+4}, \dots, L'_{j+2} L'_n, \dots, L'_{n-1} L'_n \} \\ & + \text{Span} \{ x_1, x_2, \dots, x_n \}. \end{aligned}$$

This intersection will be of $n+1-j+1$ dimensions. Since we already know $L'_{j+1}, L'_{j+2}, \dots, L'_n$ and $f'_{j+1}, f'_{j+2}, \dots, f'_{n+1}$, we already have access to $n+1-j$ dimensions, namely those established by $f'_{j+1} \circ L', f'_{j+2} \circ L', \dots, f'_{n+1} \circ L'$. Note that L' is sufficiently defined here because for each $1 \leq j \leq n-1$, f'_j , like f_j , is a function of variables x_j through x_n .

Some from the intersection defined above, it is easy to find L_j and extract $f'_j \circ L'$, which gives us f'_j .

This procedure is used on $j = n-2, j = n-3$, and so on, until $j = 2$.

For $j = 1$, we don't need to build the set constructed by set additions of the Spans defined above, we simply look at $\mathcal{F}$. After using n of the $n+1$ dimensions of $\mathcal{F}$, we have only one left. It is exactly what we need for $f'_1 \circ L'_1$ and the last component of L'_1 .

Now we have found the sequential part of F' . Inversion of the sequential part is easy to define. It is performed exactly as it would be on F , which is defined in chapter 2, section 4, the RMPKC Cryptosystem: Decryption.

5.5 Step 4: Find $\tilde{G}'$

Recall from step 1 the following representation of the public key:

$$\phi \circ P = \phi \circ \tilde{L}_3 \circ \tilde{G} \circ \tilde{L}_2 \circ F \circ L_1.$$

Now consider $\hat{G}'$ such that

$$\phi \circ P = \phi \circ \hat{G}' \circ F' \circ L'_1.$$

Our F' is not the same as F , but the span of the polynomials of $F' \circ L'_1$ is the same as the span of polynomials of $F \circ L$. In step 4, we find $\hat{G}'$, which is a straightforward exercise. After finding $\hat{G}'$, we will determine $\bar{G}'$, such that

$$\phi \circ P = \phi \circ \bar{G}' \circ \phi \circ F' \circ L'_1.$$

These maps follow our notational convention that specifies $\bar{G}'$ to be a non-homogeneous composition of a quadratic map with linear transformations, which we will show shortly.

Define

$$\hat{G}' : k^{n+1} \longrightarrow k^{n+1}, \hat{G}'(x_1, \dots, x_{n+1}) = \begin{pmatrix} \hat{g}'_1(x_1, \dots, x_{n+1}) \\ \hat{g}'_2(x_1, \dots, x_{n+1}) \\ \vdots \\ \hat{g}'_{n+1}(x_1, \dots, x_{n+1}) \end{pmatrix}$$

where

$$\forall 1 \leq i \leq n+1, \hat{g}'_i(x_1, \dots, x_{n+1}) = \sum_{j=1}^{n+1} \sum_{r=j}^{n+1} \hat{g}'_{i[j,r]} x_j x_r$$

where for all relevant values of i, j , and r , $\hat{g}'_{i[j,r]}$ is an element of k .

We find $\hat{G}'$ by using the following to identify the coefficients $\hat{g}'_{i[j,r]}$:

$$\forall 1 \leq i \leq n+1, P_i(x_1, \dots, x_n) = \sum_{j=1}^{n+1} \sum_{r=j}^{n+1} \hat{g}'_{i[j,r]} f'_j \circ L'_1(x_1, \dots, x_n) f'_r \circ L'_1(x_1, \dots, x_n).$$

By the structure of RMPKC and our development according to that structure, we know that there is a solution. We do not have a proof that in every case the solution is unique, but we have reason to believe that it is. Here is our argument: 1) any non-uniqueness that would stem from the randomness of the selection of the parameters for a specific public key is highly unlikely since the solution for the linear system of $\hat{g}'_{i[j,k]}$'s is based on $\binom{n+2}{2}$ unknowns and $\binom{n+3}{4}$ equations (because $\forall 1 \leq i \leq n+1$, $\hat{g}'_i$ is a homogeneous quadratic polynomial over $n+1$ variables and P_i is a non-homogeneous quartic polynomial in n variables); 2) any non-uniqueness that would stem from the structure of the cryptosystem would show up in our experiments, and it does not.

Finding $\hat{G}'$ was easy, and we have done it. Before proceeding to step 5 we will dehomogenize $\hat{G}'$ to form $\bar{G}'$. This exercise is also very easy. Define

$$\bar{G}' : k^n \longrightarrow k^{n+1}, \bar{G}' = \begin{pmatrix} \bar{g}'_1 \\ \bar{g}'_2 \\ \vdots \\ \bar{g}'_{n+1} \end{pmatrix}$$

where for each $1 \leq i \leq n+1$,

$$\bar{g}'_i(x_1, x_2, \dots, x_n) = \hat{g}'_i(x_1, x_2, \dots, x_n, 1).$$

Now we have successfully built L'_1, F' , and $\bar{G}'$ such that $\phi \circ P = \phi \circ \bar{G}' \circ \phi \circ F' \circ L_1$.

However, we do not have an easy way to invert $\bar{G}'$, so we still need to find a G' , an L'_3 , and an L'_2 such that G' is easy to invert. Step 5 will get us almost there, and step 6 will finish it off.

5.6 Step 5: Find $\bar{G}''$ and L'_2

Recall from step three we started with $\mathcal{F}$ and from it constructed linear transformation L'_1 and quadratic map F' in such a way that we could invert F' like any other private key and that $\mathcal{F} = \text{Span} \{ f'_i \circ L'_1 : 1 \leq i \leq n+1 \}$. Our work in step 5 is very similar. We will find linear transformation L'_2 and quadratic map $\bar{G}''$ such that $\bar{G}''$ can be inverted like in any RMPKC private key and $\text{Span} \{ \bar{g}'_i(x_1, \dots, x_n) : 1 \leq i \leq n+1 \} = \text{Span} \{ \bar{g}''_i \circ L'_2(x_1, \dots, x_n) : 1 \leq i \leq n+1 \}$.

There are two places where our work in step 5 differs from step 3. First of all, matching the $(n+1)^{st}$ component to that of the original RMPKC that we are cryptanalyzing is much easier in step 5. Secondly, we will wait until step 6 to worry about calculating parameters related to inversion of the second quadratic function in our alternate private key. This is because $\bar{G}''$ is not yet the final version which will be applied in our alternate private key.

Step 6 will be used to make the final version; at that point, it will make sense to calculate the inversion parameters. For step 5, we will only recap the parts of the process which are identical as in step 3, and we will explain fully the parts of the process which are different.

First of all, we address the core transformation. As we did in step 3, we find two components of the linear transformation L'_2 , namely $L'_{2,n-1}$ and $L'_{2,n}$ by finding the two-dimensional linear subspace of $\text{Span} \{ \bar{g}'_i : 1 \leq i \leq n+1 \}$ which is the same space as $\text{Span} \{ L'_{2,n-1}, L'_{2,n} \}$. There is also a three dimensional subspace of $\text{Span} \{ \bar{g}'_i : 1 \leq i \leq n+1 \}$ related to the core transformation, that is in $\text{Span} \{ L'_{2,n-1}L'_{2,n}, L'_{2,n-1}, L'_{2,n}, 1 \}$.

We also need to find $\bar{g}''_{n-1}$, $\bar{g}''_n$, and $\bar{g}''_{n+1}$ as we did for F' in step 3. The only difference here from step 5 is in how we find $\bar{g}''_{n+1}$. In step 3, finding f'_{n+1} was much harder. We had to use the trick of finding the subspace of $\text{Span} \{ p_i : 1 \leq i \leq n+1 \}$ which, as a polynomial, is divisible by $f'_{n+1} \circ L'_1$ as a polynomial.

For $\bar{g}''_{n+1}$, it is much easier. We have $\bar{g}'_{n+1}$ and we force

$$\bar{g}''_{n+1}(L'_{(2,n-1)}, L'_{(2,n)}) = \bar{g}'_{n+1}(x_1, \dots, x_n, 1),$$

where $L'_{(2,n-1)}$ and $L'_{(2,n)}$ are expressed as functions of $x_1, \dots, x_n$. The linear algebra required to perform this operation is straightforward. Determining $\bar{g}'_{n-1}$ and $\bar{g}'_n$ is the same as in step 3.

Finding the sequential part of $\bar{G}''$ also follows exactly as in step 3. The iterative process determines, in order, $L'_{2,i}$ and $\bar{g}''_i$ for $i = n-2, n-1, \dots, 1$. At this point we have fully determined both $\bar{G}''$ and L'_2 such that $\text{Span} \{ \bar{g}'_i(x_1, \dots, x_n, 1) : 1 \leq i \leq n+1 \} = \text{Span} \{ \bar{g}''_i \circ L'_2(x_1, \dots, x_n) : 1 \leq i \leq n+1 \}$. A simple linear algebra exercise is then used to find $\hat{L}'_3$ such that $\hat{L}'_3 \circ \bar{G}'' \circ L'_2 = \bar{G}'$.

5.7 Step 6: Find L'_3 and G'

From step 3, we have L'_1 and F' . From step 5, we have L'_2 , $\bar{G}''$, and $\bar{L}'_3$. The construction of these maps was done in a way such that $P = \bar{L}'_3 \circ \bar{g}''_i \circ L'_2 \circ F' \circ L'_1$. Furthermore, L'_1 and L'_2 are linear transformations and F' and $\bar{G}''$ are structured as the quadratic functions of an RMPKC. However, $\bar{L}'_3$ is a linear transformation from k^{n+1} to k^{n+1} and we need a linear transformation from k^n to k^n .

Step 6 will give us L'_3 and G' where L'_3 is a linear transformation from k^n to k^n , G' can be inverted as any other quadratic map in an RMPKC, and $\phi \circ \bar{L}'_3 \circ \bar{G}'' = L'_3 \circ \phi \circ G'$, which results in

$$\phi \circ P = L'_3 \circ \phi \circ G \circ L_2 \circ \phi \circ F \circ L_1 = L'_3 \circ \phi \circ G' \circ L'_2 \circ \phi \circ F' \circ L'_1.$$

To accomplish this step, we will define two linear transformations, namely $\tilde{L}'_3$ and π such that $\bar{L}'_3 = \tilde{L}'_3 \pi$. Furthermore, $\tilde{L}'_3$ will be constructed such that each of its first n components are independent of the $(n+1)^{st}$ component of the domain space and its $(n+1)^{st}$ component is independent of the first n components of the domain space. Actually, this $(n+1)^{st}$

component of $\tilde{L}'_3$ will be trivial. It transforms $\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n+1} \end{pmatrix}$ into (x_{n+1}) . This will enable us to

define L'_3 such that $\phi \circ L'_3 = \tilde{L}'_3 \circ \phi$. Additionally, π must be upper triangular so that $\pi \circ \bar{G}''$ is in the form appropriate for an RMPKC. This will enable us to define a means to invert $\pi \circ \bar{G}''$, which we will name G' .

To find valid transformations $\tilde{L}'_3$ and π which meet our requirements is not hard. We first find the inverse of $\tilde{L}'_3$, namely $(\tilde{L}'_3)^{-1}$, and then find its inverse, which will be $\tilde{L}'_3$. The specific construction of $\tilde{L}'_3$ as noted above must also apply to $(\tilde{L}'_3)^{-1}$.

To see how we find $(\tilde{L}'_3)^{-1}$ it will be easier using matrices to represent the linear transfor-

mations. $\bar{L}'_3$ is given to us. $\bar{L}'_3$ is of the form $\begin{pmatrix} * & \cdots & * & * \\ \vdots & \ddots & \vdots & \vdots \\ * & \cdots & * & * \\ 0 & \cdots & 0 & * \end{pmatrix}$; the zeros in the bottom row

are there because of the way we specifically defined $\bar{g}''_{n+1}$ from $\bar{g}''_{n+1}$. $(\tilde{L}'_3)^{-1}$ must of the form $\begin{pmatrix} * & \cdots & * & 0 \\ \vdots & \ddots & \vdots & \vdots \\ * & \cdots & * & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix}$ and thus has n^2 coefficients for us to determine. π is upper triangular, with $\frac{n(n+1)}{2}$ unknown coefficients and $\frac{n(n-1)}{2}$ known coefficients, each equal to 0.

Consider the matrix equation $(\tilde{L}'_3)^{-1}\bar{L}'_3 = \pi$. $(\tilde{L}'_3)^{-1}\bar{L}'_3$ is an $(n+1) \times (n+1)$ matrix where each element is a linear combination of the unknown elements of $(\tilde{L}'_3)^{-1}$. We care to ensure that each of those elements of $(\tilde{L}'_3)^{-1}\bar{L}'_3$ are equal to the corresponding elements of π . The upper triangular elements of π can be anything. Each of the sub-diagonal elements of π must be zero.

So we build a system of linear equations based on these sub-diagonal elements, from $(\tilde{L}'_3)^{-1}\bar{L}'_3 = \pi$. There are $\frac{n(n-1)}{2}$ equations and n^2 unknowns (the elements of $(\tilde{L}'_3)^{-1}$). For any positive value of n , we have more unknowns than equations; so we will either get an inconsistent system or multiple solutions. Since we constructed our system based upon the RMPKC, we always get a solution. Furthermore, we never have trouble finding a solution which leads to an invertible $(\tilde{L}'_3)^{-1}$. From $(\tilde{L}'_3)^{-1}$ we calculate $\tilde{L}'_3$, which is of its desired form. We also calculate $\pi = (\tilde{L}'_3)^{-1}\bar{L}'_3$, which also is of its desired form.

What is left is to find G' and the key to the inversion of G' . From our work described above, we have $\bar{L}'_3 \circ \bar{G}'' = \tilde{L}'_3 \circ \pi \circ \bar{G}''$. So let's define $G' = \pi \circ \bar{G}''$, which gives us $P = \bar{L}'_3 \circ G' \circ L'_2 \circ \phi \circ F' \circ L'_1$. We also get G' in the form of a quadratic map of RMPKC. This is true because π is upper triangular, which makes for each $i = 1, 2, \dots, n+1$, g'_i a

linear combination of $\bar{g}_i'', \bar{g}_{i+1}'', \dots, \bar{g}_{n+1}''$. The structure of the quadratic map of RMPKC is preserved with these linear combinations.

We are almost there. Since we built $\tilde{L}'_3$ in the form $\begin{pmatrix} * & \cdots & * & 0 \\ \vdots & \ddots & \vdots & \vdots \\ * & \cdots & * & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix}$, we are ready to define $L'_3 : k^n \longrightarrow k^n$ as the same linear transformation as the first n components of the linear transformation $\tilde{L}'_3$. Since $\phi \circ \tilde{L}'_3 = L'_3 \circ \phi$, we finally have

$$\phi \circ P = \phi \circ \tilde{L}'_3 \circ \pi \circ \bar{G}'' \circ L'_2 \circ F' \circ L'_1 = L'_3 \circ \phi \circ G' \circ L'_2 \circ \phi \circ F' \circ L'_1.$$

The last step of our cryptanalysis is to figure out how to invert G' . Fortunately, we figure this out exactly in the same manner that we figured out how to invert F' . The core transformation polynomials, G'_{n-1}, G'_n , and G'_{n+1} , are used to solve for G' 's core transformation inversion function parameters $(\delta'_1, \dots, \delta'_6, \gamma'_1, \dots, \gamma'_6)$ which correspond to F' 's core transformation inversion function parameters $(\alpha'_1, \dots, \alpha'_6, \beta'_1, \dots, \beta'_6)$.

Inversion of the sequential part of G' is performed just like the sequential part of F' (which is the same as inverting the sequential part of F and G , when you have access to the private key).

At this point our cryptanalysis is complete. We have take a public key, $\phi \circ P$, which we know was constructed from a composition of some hidden maps, $\phi \circ P = L_3 \circ \phi \circ G \circ L_2 \circ \phi \circ F \circ L_1$, and constructed our own decomposition, namely $\phi \circ P = L'_3 \circ \phi \circ G' \circ L'_2 \circ \phi \circ F' \circ L'_1$ and the means to invert the public key exactly as successfully as the owner of the private key would do using the private key.

5.8 Small-Scale Example

We here demonstrate the cryptanalytic process with a small-scale example. We will use the same example employed in chapter 2, which demonstrated the construction of the public

key and the encryption and decryption processes. The example we have chosen enables all calculations to be done by hand. However, it has two drawbacks.

Firstly, using a field of odd characteristic prevents the solution of the quadratic equation by the method utilized in the research. In our example, however, observation of the system of equations immediately presents a solution to us.

Secondly, all five maps, which usually get coefficients chosen via some random process, are loaded with coefficients equally zero, which make the hand calculations sane. However, the resulting extra simple public key disables our method to determine f'_{n+1} by finding a subspace of $Span \{ p_i : 1 \leq i \leq n+1 \}$ which it divides. In our case, we cheat and set $f'_{n+1} = f_{n+1}$, which we wouldn't have access to in a true scenario.

The rest of the cryptanalysis follows directly from what is described in the earlier sections of this chapter.

Recall that the public key for our example utilizes $k = GF(11)$, $n = 3$, and the following map:

$$P(x_1, x_2, x_3) = \begin{pmatrix} 2x_1^2x_2^2 + 2x_1x_2^2x_3 + 2x_1x_2x_3 + 3x_1x_2^2 \\ 2x_1x_2x_3 + 2x_2x_3^2 \\ 2x_2x_3 \\ 3x_1x_2^2 + 3x_2^2x_3 \end{pmatrix}.$$

Our first step in the cryptanalysis is to homogenize P . So let

$$\tilde{P}(x_1, x_2, x_3, x_4) = \begin{pmatrix} 2x_1^2x_2^2 + 2x_1x_2^2x_3 + 2x_1x_2x_3x_4 + 3x_1x_2^2x_4 \\ 2x_1x_2x_3x_4 + 2x_2x_3^2x_4 \\ 2x_2x_3x_4^2 \\ 3x_1x_2^2x_4 + 3x_2^2x_3x_4 \end{pmatrix}.$$

Step 2 applies the enhanced partial derviative attack as follows. The order of the partial derivatives enables us to more readily see a basis for the space spanned by the partial

derivatives. The first set listed enable direct identification of a few monomial basis elements.

$$\frac{\partial p_4(x_1, x_2, x_3, x_4)}{\partial x_1} = 3x_2^2 x_4$$

$$\frac{\partial p_2(x_1, x_2, x_3, x_4)}{\partial x_1} = 2x_2 x_3 x_4$$

$$\frac{\partial p_3(x_1, x_2, x_3, x_4)}{\partial x_3} = 2x_2 x_4^2$$

$$\frac{\partial p_3(x_1, x_2, x_3, x_4)}{\partial x_2} = 2x_3 x_4^2$$

yield $\{ x_2^2 x_4, x_2 x_3 x_4, x_2 x_4^2, x_3 x_4^2 \}$ to begin with. Now let's consider

$$\frac{\partial p_2(x_1, x_2, x_3, x_4)}{\partial x_3} = 2x_1 x_2 x_4 + 4x_2 x_3 x_4,$$

which increases our basis building to $\{ x_1 x_2 x_4, x_2^2 x_4, x_2 x_3 x_4, x_2 x_4^2, x_3 x_4^2 \}$. So we add

$$\frac{\partial p_1(x_1, x_2, x_3, x_4)}{\partial x_3} = 2x_1 x_2^2 + 2x_1 x_2 x_4,$$

which yields $\{ x_1 x_2^2, x_1 x_2 x_4, x_2^2 x_4, x_2 x_3 x_4, x_2 x_4^2, x_3 x_4^2 \}$. We now consider

$$\frac{\partial p_1(x_1, x_2, x_3, x_4)}{\partial x_4} = 2x_1 x_2 x_3 + 3x_1 x_2^2,$$

$$\frac{\partial p_4(x_1, x_2, x_3, x_4)}{\partial x_4} = 3x_1 x_2^2 + 3x_2^2 x_3,$$

which gives us $\{ x_1 x_2^2, x_1 x_2 x_3, x_1 x_2 x_4, x_2^2 x_3, x_2^2 x_4, x_2 x_3 x_4, x_2 x_4^2, x_3 x_4^2 \}$. Our last chance to add a monomial to the basis comes from

$$\frac{\partial p_2(x_1, x_2, x_3, x_4)}{\partial x_4} = 2x_1 x_2 x_3 + 2x_2 x_3^2,$$

yielding $\{ x_1 x_2^2, x_1 x_2 x_3, x_1 x_2 x_4, x_2^2 x_3, x_2 x_3^2, x_2^2 x_4, x_2 x_3 x_4, x_2 x_4^2, x_3 x_4^2 \}$. Two binomial basis elements are required by

$$\frac{\partial p_1(x_1, x_2, x_3, x_4)}{\partial x_2} = 4x_1^2 x_2 + 4x_1 x_2 x_3 + 2x_1 x_3 x_4 + 6x_1 x_2 x_4$$

$$\frac{\partial p_2(x_1, x_2, x_3, x_4)}{\partial x_2} = 2x_1 x_3 x_4 + 2x_3^2 x_4,$$

which give us a final basis for the span of the partial derivatives:

$$\{ x_1x_2^2, x_1x_2x_3, x_1x_2x_4, x_2^2x_3, x_2x_3^2, x_2^2x_4, x_2x_3x_4, \\ x_2x_4^2, x_3x_4^2, x_1x_3x_4 + 2x_1^2x_2, x_1x_3x_4 + x_3^2x_4 \}.$$

All of the other partial derivatives are linear combinations of the monomial basis elements.

Per our presentation of the cryptanalysis, we let $D =$

$$Span \{ x_1x_2^2, x_1x_2x_3, x_1x_2x_4, x_2^2x_3, x_2x_3^2, x_2^2x_4, x_2x_3x_4, \\ x_2x_4^2, x_3x_4^2, x_1x_3x_4 + 2x_1^2x_2, x_1x_3x_4 + x_3^2x_4 \}$$

and define

$$R = \{ \theta : \forall 1 \leq i \leq n+1, x_i \theta \in D \}.$$

For our hand calculations, we shall use

$$\forall 1 \leq i \leq n+1, \bar{R}_i = \{ \theta : x_i \theta \in D \}, \quad R = \bigcap_{i=1}^{n+1} \bar{R}_i$$

forming each $\bar{R}_i$ by inspection of D .

$$\bar{R}_1 = Span \{ x_2^2, x_2x_3, x_2x_4, x_3x_4 + 2x_1x_2 \}$$

$$\bar{R}_2 = Span \{ x_1x_2, x_1x_3, x_1x_4, x_2x_3, x_2x_4, x_3^2, x_3x_4, x_4^2 \}$$

$$\bar{R}_3 = Span \{ x_1x_2, x_2^2, x_2x_3, x_2x_4, x_4^2, x_1x_4 + x_3x_4 \}$$

$$\bar{R}_4 = Span \{ x_1x_2, x_2^2, x_2x_3, x_2x_4, x_3x_4, x_1x_3 + x_3^2 \}$$

$$R = Span \{ x_2x_3, x_2x_4 \}$$

Here the direct application of the partial derivative attack fails. So we will use the enhanced version developed in our research. We start by defining and calculating

$$\forall 1 \leq s \leq n+1, D_s = Span \left\{ \frac{\partial}{\partial x_j} \tilde{p}_i(x_1, \dots, x_{s-1}, 0, x_{s+1}, \dots, x_{n+1}) : 1 \leq i, j, \leq n+1 \right\}.$$

We can find D_s for each value of s by evaluating each of the polynomials that define the basis elements of D at $x_s = 0$.

$$D_1 = \text{Span} \{ x_2^2 x_3, x_2 x_3^2, x_2^2 x_4, x_2 x_3 x_4, x_2 x_4^2, x_3 x_4^2, x_3^2 x_4 \}$$

$$D_2 = \text{Span} \{ x_3 x_4^2, x_1 x_3 x_4, x_3^2 x_4 \}$$

$$D_3 = \text{Span} \{ x_1 x_2^2, x_1 x_2 x_4, x_2^2 x_4, x_2 x_4^2, 2x_1^2 x_2 \}$$

$$D_4 = \text{Span} \{ x_1 x_2^2, x_1 x_2 x_3, x_2^2 x_3, x_2 x_3^2, 2x_1^2 x_2 \}$$

Note that the last two monomial basis elements listed for D_2 relied on both binomial basis elements in D . Now calculating R_s for $s = 1, 2, 3, 4$ by inspection is not too hard.

$$R_1 = \text{Span} \{ x_2 x_3, x_2 x_4, x_3 x_4 \}$$

$$R_2 = \text{Span} \{ x_3 x_4 \}$$

$$R_3 = \text{Span} \{ x_1 x_2, x_2 x_4 \}$$

$$R_4 = \text{Span} \{ x_1 x_2, x_2 x_3 \}$$

In the cryptanalysis explained earlier in this chapter, we would then calculate for each $s = 1, 2, 3, 4$,

$$R_x^+ = R_x + \text{Span} \{ x_s x_i : 1 \leq i \leq n+1 \}$$

and then calculate $R^+ = \bigcap_{s=1}^{n+1} R_s^+$. However in our small-scale example, it will be easier to consider each of the basis elements in R_s for $s = 1, 2, 3, 4$ individually.

Firstly, $x_1 x_2$ is in each of R_3 and R_4 , and also in each of R_1^+ and R_2^+ ; thus $x_1 x_2 \in R^+$. Secondly, $x_2 x_3$ is in each of R_1 and R_4 , as well as R_2^+ and R_3^+ . Also, $x_2 x_4$ is in R_1 and R_3 , as well as R_2^+ and R_4^+ . Lastly, $x_3 x_4$ is in each of R_1, R_2, R_3^+ , and R_4^+ . That gives us the four dimensional space $R^+ = \text{Span} \{ x_1 x_2, x_2 x_3, x_2 x_4, x_3 x_4 \}$. At this point we dehomogenize R^+ and define $\mathcal{F} = \text{Span} \{ x_1 x_2, x_2 x_3, x_2, x_3 \}$. We have completed step 2 of our cryptanalysis.

In step 3, we determine F' and L'_1 such that $\text{Span} \{ f'_i \circ L'_1(x_1, x_2, x_3, x_4) : 1 \leq i \leq n+1 \} = \mathcal{F}$ and we can use the inversion technique of RMPKC on F' . We start by examining $\mathcal{F}$ to find a two-dimensional linear subspace. We see the basis elements x_2 and x_3 . We look for a homogeneous quadratic polynomial in $\text{Span} \{ x_2^2, x_2x_3, x_3^2 \}$ and immediately see x_2x_3 . So we will want to make $\text{Span} \{ f'_2(x_1, x_2, x_3, x_4), f'_3(x_1, x_2, x_3, x_4), f'_4(x_1, x_2, x_3, x_4) \} = \text{Span} \{ x_2, x_3, x_2x_3 \}$. At this point we can let

$$L'_{1,2}(x_1, x_2, x_3, x_4) = x_2, \quad L'_{1,3}(x_1, x_2, x_3, x_4) = x_3$$

since our quadratic component of the core transformation is already x_2x_3 . This is the point where we get around our potential problem of using a field with odd characteristic, because we do not have to solve a quadratic equation to find $L'_{1,2}$ and $L'_{1,3}$.

The second problem we mentioned with our small-scale case is finding f'_{n+1} exactly. We cheat here and set $f'_4 = x_2$. To make the formulae for each of f'_2 and f'_3 easy, we choose $f'_2(x_1, x_2, x_3, x_4) = x_3$ and $f'_3(x_1, x_2, x_3, x_4) = x_2x_3$.

Our next task is to find $L'_{1,1}$ and f'_1 . Since $L'_{1,2}(x_1, x_2, x_3) = x_2$ and $L'_{1,3}(x_1, x_2, x_3) = x_3$, let $L'_{1,1}(x_1, x_2, x_3) = x_1$. Let's also use $f'_1(x_1, x_2, x_3) = x_1x_2$, which gives us $\text{Span} \{ f'_i \circ L'_1(x_1, x_2, x_3) : 1 \leq i \leq n+1 \} = \mathcal{F}$. We need a way to invert F' , so we find solutions for our α_i and β'_i for each $i = 1, 2, \dots, 6$.

$$x_2 = \frac{f'_3(x_1, x_2, x_3)\beta'_1 + f'_2(x_1, x_2, x_3)\beta'_2 + f'_4(x_1, x_2, x_3)\beta'_3}{f'_3(x_1, x_2, x_3)\beta'_4 + f'_2(x_1, x_2, x_3)\beta'_5 + f'_4(x_1, x_2, x_3)\beta'_6}$$

$$x_2 = \frac{x_2x_3\beta'_1 + x_3\beta'_2 + x_2\beta'_3}{x_2x_3\beta'_4 + x_3\beta'_5 + x_2\beta'_6}$$

which yields

$$x_2^2x_3\beta'_4 + x_2x_3\beta'_5 + x_2^2\beta'_6 = x_2x_3\beta'_1 + x_3\beta'_2 + x_2\beta'_3$$

forcing $\beta'_1 = \beta'_5$ and $\beta'_2 = \beta'_3 = \beta'_4 = \beta'_6 = 0$. For simplicity, let's choose $\beta'_1 = \beta'_5 = 1$.

$$x_3 = \frac{f'_3(x_1, x_2, x_3)\alpha'_1 + f'_2(x_1, x_2, x_3)\alpha'_2 + f'_4(x_1, x_2, x_3)\alpha'_3}{f'_3(x_1, x_2, x_3)\alpha'_4 + f'_2(x_1, x_2, x_3)\alpha'_5 + f'_4(x_1, x_2, x_3)\alpha'_6}$$

$$x_2 = \frac{x_2x_3\alpha'_1 + x_3\alpha'_2 + x_2\alpha'_3}{x_2x_3\alpha'_4 + x_3\alpha'_5 + x_2\alpha'_6}$$

which yields

$$x_2x_3^2\alpha'_4 + x_3^2\alpha'_5 + x_2x_3\alpha'_6 = x_2x_3\alpha'_1 + x_3\alpha'_2 + x_2\alpha'_3$$

forcing $\alpha'_1 = \alpha'_6 = 1$ and $\alpha'_2 = \alpha'_3 = \alpha'_4 = \alpha'_5 = 0$. Again we'll choose for simple calculations and let $\alpha'_1 = \alpha'_6 = 1$.

We now have defined L'_1, F' and the means to invert F' . Since $L'_1 = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$, we get

$(L'_1)^{-1} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$. This completes step 3.

In step 4, we find $\hat{G}'$. The general form for each component polynomial of $\hat{G}'$ is the following (for $i = 1, 2, 3, 4$):

$$\begin{aligned} \hat{g}'_i(w_1, w_2, w_3, w_4) = & \hat{g}'_{i[1,1]}w_1^2 + \hat{g}'_{i[1,2]}w_1w_2 + \hat{g}'_{i[1,3]}w_1w_3 + \hat{g}'_{i[1,4]}w_1w_4 + \hat{g}'_{i[2,2]}w_2^2 \\ & + \hat{g}'_{i[2,3]}w_2w_3 + \hat{g}'_{i[2,4]}w_2w_4 + \hat{g}'_{i[3,3]}w_3^2 + \hat{g}'_{i[3,4]}w_3w_4 + \hat{g}'_{i[4,4]}w_4^2 \end{aligned}$$

For the polynomials associated with $\hat{G}' \circ F' \circ L'_1$, we get

$$\begin{aligned} \hat{g}'_i \circ F' \circ L'_1(x_1, x_2, x_3) = & \hat{g}'_{i[1,1]}(x_1x_2)^2 + \hat{g}'_{i[1,2]}(x_1x_2)(x_3) + \hat{g}'_{i[1,3]}(x_1x_2)(x_2x_3) + \hat{g}'_{i[1,4]}(x_1x_2)(x_2) \\ & + \hat{g}'_{i[2,2]}(x_3)^2 + \hat{g}'_{i[2,3]}(x_3)(x_2x_3) + \hat{g}'_{i[2,4]}(x_3)(x_2) + \hat{g}'_{i[3,3]}(x_2x_3)^2 + \hat{g}'_{i[3,4]}(x_2x_3)(x_2) + \hat{g}'_{i[4,4]}(x_2)^2 \end{aligned}$$

For $i = 1$ this polynomial must equal $2x_1^2x_2^2 + 2x_1x_2^2x_3 + 2x_1x_2x_3 + 3x_1x_2^2$ so it must be that $\hat{g}'_1(x_1, x_2, x_3, x_4) = 2x_1x_2 + x_1x_3 + 3x_1$.

For $i = 2$ this polynomial must equal $2x_1x_2x_3 + 2x_2x_3^2$ so we must have $\hat{g}'_2(x_1, x_2, x_3, x_4) = x_2x_3$.

For $i = 3$ this polynomial must equal $2x_2x_3$ so we get $\hat{g}'_3(x_1, x_2, x_3, x_4) = x_3$.

Finally for $i = 4$ this polynomial must equal $3x_1x_2^2 + 3x_2^2x_3$ which gives us

$$\hat{g}'_4(x_1, x_2, x_3, x_4) = 3x_2.$$

We dehomogenize $\hat{G}'$ into $\bar{G}'$.

$$\bar{G}'(x_1, x_2, x_3, x_4) = \begin{pmatrix} 2x_1^2 + 2x_1x_2 + 2x_1x_3 + 3x_1 \\ 2x_1x_2 + 2x_2x_3 \\ 2x_2 \\ 3x_1 + 3x_3 \end{pmatrix}$$

This completes step 4 of the cryptanalysis. In step 5, we need to find $\bar{G}''$, L'_2 , and $\hat{L}'_3$ such that $\bar{G}' = \hat{L}'_3 \circ \bar{G}'' \circ L'_2$, $\bar{G}''$ is structured like the G of RMPKC, and L'_2 , and $\hat{L}'_3$ are homogeneous linear maps. Remember step 5 works very similarly to step 3.

We start by finding a two-dimensional subspace of $\text{Span} \{ \bar{g}''_i(x_1, x_2, x_3) : 1 \leq i \leq n+1 \}$. By inspection we see $\text{Span} \{ x_2, x_1 + x_3 \}$. We also see that $x_2(x_1 + x_3) \in \text{Span} \{ \bar{g}''_i(x_1, x_2, x_3) : 1 \leq i \leq n+1 \}$, so we can use $L'_{2,2} = x_2$ and $L'_{2,3} = x_1 + x_3$. By inspection we get $\bar{g}''_2(x_1, x_2, x_3) = 2x_2x_3$, $\bar{g}''_3(x_1, x_2, x_3) = 2x_2$, and $\bar{g}''_4(x_1, x_2, x_3) = 3x_3$. We arbitrarily pick $L'_{2,1} = x_3$ to make L'_2 simple yet invertible.

While we are at it, let's find the inverse of $L'_2(x_1, x_2, x_3) = \begin{pmatrix} x_3 \\ x_2 \\ x_1 + x_3 \end{pmatrix}$, which gives us

$$(L'_2)^{-1}(x_1, x_2, x_3) = \begin{pmatrix} 10x_1 + x_3 \\ x_2 \\ x_1 \end{pmatrix}. \text{ What is left is to find } \bar{g}''_1.$$

The general form for $\bar{g}''_1$ is

$$\begin{aligned} \hat{g}''_1(w_1, w_2, w_3) = & \hat{g}''_{1[1,1]}w_1^2 + \hat{g}''_{1[1,2]}w_1w_2 + \hat{g}''_{1[1,3]}w_1w_3 + \hat{g}''_{1[1,4]}w_1 + \hat{g}''_{1[2,2]}w_2^2 \\ & + \hat{g}''_{1[2,3]}w_2w_3 + \hat{g}''_{1[2,4]}w_2 + \hat{g}''_{1[3,3]}w_3^2 + \hat{g}''_{1[3,4]}w_3 + \hat{g}''_{1[4,4]}. \end{aligned}$$

By composing $\bar{g}_1''$ with L_2' we get

$$\begin{aligned}\hat{g}_1'' \circ L_2'(x_1, x_2, x_3) &= \hat{g}_{1[1,1]}'' x_3^2 + \hat{g}_{1[1,2]}'' x_2 x_3 + \hat{g}_{1[1,3]}'' x_1 x_3 + x_3^2 + \hat{g}_{1[1,4]}'' x_3 + \hat{g}_{1[2,2]}'' x_2^2 \\ &+ \hat{g}_{1[2,3]}'' [x_1 x_2 + x_2 x_3] + \hat{g}_{1[2,4]}'' x_2 + \hat{g}_{1[3,3]}'' [x_1^2 + 2x_1 x_3 + x_3^2] + \hat{g}_{1[3,4]}'' [x_1 + x_3] + \hat{g}_{1[4,4]}'',\end{aligned}$$

which must be equal to $\bar{g}'(x_1, x_2, x_3) = 2x_1^2 + 2x_1 x_3 + 2x_1 x_2 + 3x_1$.

The polynomial of $\hat{g}_1'' \circ L_2'(x_1, x_2, x_3)$ will equal the polynomial of $\bar{g}'(x_1, x_2, x_3)$, term by term. That gives us 10 equations and 10 unknowns. We will list these equations in an order to show how we can solve the system by hand. In each case, the monomial will specify the term involved, followed by the resulting equation for the coefficient of that term. In most cases, the equation indicates directly the solution. Where appropriate, an arrow to the right will indicate "implies", followed by the solution of the equation using substitution of previously solved coefficients of $\bar{g}_1''$.

$$\begin{aligned}x_2^2 : \bar{g}_{1[2,2]}'' &= 0 & x_1 x_2 : \bar{g}_{1[2,3]}'' &= 2 & x_2 : \bar{g}_{1[2,4]}'' &= 0 & x_1^2 : \bar{g}_{1[3,3]}'' &= 2 \\ x_1 : \bar{g}_{1[3,4]}'' &= 3 & 1 : \bar{g}_{1[4,4]}'' &= 0 & x_2 x_3 : \bar{g}_{1[1,2]}'' + \bar{g}_{1[2,3]}'' &= 0 = \bar{g}_{1[1,2]}'' + 2 & \longrightarrow & \bar{g}_{1[1,2]}'' = 9 \\ x_1 x_3 : \bar{g}_{1[1,3]}'' + 2\bar{g}_{1[3,3]}'' &= 2 = \bar{g}_{1[1,3]}'' + 2(2) & \longrightarrow & \bar{g}_{1[1,3]}'' = 9 \\ x_3 : \bar{g}_{1[1,4]}'' + \bar{g}_{1[3,4]}'' &= 0 = \bar{g}_{1[1,2]}'' + 3 & \longrightarrow & \bar{g}_{1[1,4]}'' = 8 \\ x_3^2 : \bar{g}_{1[1,1]}'' + \bar{g}_{1[3,3]}'' + \bar{g}_{1[1,3]}'' &= 0 = \bar{g}_{1[1,1]}'' + 2 + 9 & \longrightarrow & \bar{g}_{1[1,1]}'' = 0\end{aligned}$$

Our selection for $\bar{G}''$ is complete.

$$\bar{G}''(x_1, x_2, x_3) = \begin{pmatrix} 9x_1 x_2 + 9x_1 x_3 + 8x_1 + 2x_2 x_3 + 2x_3^2 + 3x_3 \\ 2x_2 x_3 \\ 2x_2 \\ 3x_3 \end{pmatrix}$$

Furthermore $\bar{G}'' \circ L'_2 = \bar{G}'$, so in this case we have $\hat{L}'_3(x_1, x_2, x_3, x_4) = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$ and thus

$$L'_3(x_1, x_2, x_3) = (L'_3(x_1, x_2, x_3))^{-1} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}. \text{ We let } G' = \bar{G}'' \text{ and the construction of an}$$

alternate formulation of the public key is complete. All is left of step 6 is to determine the inversion parameters $\delta'_1, \delta'_2, \dots, \delta'_6$ and $\gamma'_1, \gamma'_2, \dots, \gamma'_6$. We use again the method shown in step 3.

$$x_2 = \frac{g'_3(x_1, x_2, x_3)\gamma'_1 + g'_2(x_1, x_2, x_3)\gamma'_2 + g'_4(x_1, x_2, x_3)\gamma'_3}{g'_3(x_1, x_2, x_3)\gamma'_4 + g'_2(x_1, x_2, x_3)\gamma'_5 + g'_4(x_1, x_2, x_3)\gamma'_6}$$

$$x_2 = \frac{2x_2\gamma'_1 + 2x_2x_3\gamma'_2 + 3x_3\gamma'_3}{2x_2\gamma'_4 + 2x_2x_3\gamma'_5 + 3x_3\gamma'_6}$$

which yields

$$2x_2^2\gamma'_4 + 2x_2^2x_3\gamma'_5 + 3x_2x_3\gamma'_6 = 2x_2\gamma'_1 + 2x_2x_3\gamma'_2 + 3x_3\gamma'_3$$

and thus

$$\gamma'_1 = \gamma'_3 = \gamma'_4 = \gamma'_5 = 0, \quad 2\gamma'_2 = 3\gamma'_6$$

so we choose $\gamma'_2 = 1$ and $\gamma'_6 = 8$. Now for the δ 's.

$$x_3 = \frac{g'_3(x_1, x_2, x_3)\delta'_1 + g'_2(x_1, x_2, x_3)\delta'_2 + g'_4(x_1, x_2, x_3)\delta'_3}{g'_3(x_1, x_2, x_3)\delta'_4 + g'_2(x_1, x_2, x_3)\delta'_5 + g'_4(x_1, x_2, x_3)\delta'_6}$$

$$x_3 = \frac{2x_2\delta'_1 + 2x_2x_3\delta'_2 + 3x_3\delta'_3}{2x_2\delta'_4 + 2x_2x_3\delta'_5 + 3x_3\delta'_6}$$

which yields

$$2x_2x_3\delta'_4 + 2x_2x_3^2\delta'_5 + 3x_3^2\delta'_6 = 2x_2\delta'_1 + 2x_2x_3\delta'_2 + 3x_3\delta'_3$$

therefore, $\delta'_1 = \delta'_3 = \delta'_5 = \delta'_6 = 0$ and $2\delta'_4 = 2\delta'_2$, so we let $\delta'_4 = \delta'_2 = 1$.

Our cryptanalysis is complete. Here is a summary of our alternative private key:

$$\begin{aligned}
(L'_1)^{-1}(x_1, x_2, x_3) &= \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, (L'_2)^{-1}(x_1, x_2, x_3) = \begin{pmatrix} 10x_1 + x_3 \\ x_2 \\ x_1 \end{pmatrix}, (L'_3)^{-1}(x_1, x_2, x_3) = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \\
F'(x_1, x_2, x_3) &= \begin{pmatrix} x_1x_2 \\ x_3 \\ x_2x_3 \\ x_2 \end{pmatrix}, (\alpha'_1, \alpha'_2, \dots, \alpha'_6) = (1, 0, 0, 0, 0, 1), (\beta'_1, \beta'_2, \dots, \beta'_6) = (1, 0, 0, 0, 1, 0) \\
G'(x_1, x_2, x_3) &= \begin{pmatrix} 9x_1x_2 + 9x_1x_3 + 8x_1 + 2x_2x_3 + 2x_3^2 + 3x_3 \\ 2x_2x_3 \\ 2x_2 \\ 3x_3 \end{pmatrix} \\
(\delta'_1, \delta'_2, \dots, \delta'_6) &= (0, 1, 0, 1, 0, 0), (\gamma'_1, \gamma'_2, \dots, \gamma'_6) = (0, 1, 0, 0, 0, 8)
\end{aligned}$$

Let us test inversion of a ciphertext. We will use the same cipher text generated previously for demonstration of our small-scale example. $Y = \begin{pmatrix} 2 \\ 7 \\ 2 \end{pmatrix}$. Since $(L'_3)^{-1}$ is the identity matrix,

$(L'_3)^{-1}(2, 7, 2) = \begin{pmatrix} 2 \\ 7 \\ 2 \end{pmatrix}$. Our next step is to apply the inversion of the core transformation

to $\begin{pmatrix} 2 \\ 7 \\ 2 \end{pmatrix}$. Since $(\gamma'_1, \gamma'_2, \dots, \gamma'_6) = (0, 1, 0, 0, 0, 8)$ and $(\delta'_1, \delta'_2, \dots, \delta'_6) = (0, 1, 0, 1, 0, 0)$,

$$\begin{pmatrix} 2 \\ 7 \\ 2 \end{pmatrix} \rightarrow \begin{pmatrix} \frac{\gamma'_1(2) + \gamma'_2(7) + \gamma'_3}{\gamma'_4(2) + \gamma'_5(7) + \gamma'_6} \\ \frac{\delta'_1(2) + \delta'_2(7) + \delta'_3}{\delta'_4(2) + \delta'_5(7) + \delta'_6} \end{pmatrix} = \begin{pmatrix} \frac{(0)(2) + (1)(7) + (0)}{(0)(2) + (0)(7) + (8)} \\ \frac{(0)(2) + (1)(7) + (0)}{(1)(2) + (0)(7) + (0)} \end{pmatrix} = \begin{pmatrix} \frac{7}{8} \\ \frac{7}{2} \end{pmatrix} = \begin{pmatrix} 5 \\ 9 \end{pmatrix}$$

For the sequential part of G' , we solve $\frac{g'_1(x_1, x_2, x_3)}{g'_4(x_1, x_2, x_3)} = 2 = \frac{9x_1x_2 + 9x_1x_3 + 8x_1 + 2x_2x_3 + 2x_3^2 + 3x_3}{3x_3}$ for

unknown x_1 and known $x_2 = 5, x_3 = 9$.

$$\begin{aligned}\frac{g'_1(x_1, 5, 9)}{g'_4(x_1, 5, 9)} &= \frac{9x_1(5) + 9x_1(9) + 8x_1 + 2(5)(9) + 2(9)^2 + 3(9)}{3(9)} \\ &= \frac{x_1[1 + 4 + 8] + 2 + 8 + 5}{5} = \frac{2x_1 + 4}{5} = 2\end{aligned}$$

which yields $x_1 = \frac{2(5)-4}{2} = \frac{6}{2} = 3$. Thus the inversion of G' on $\begin{pmatrix} 2 \\ 7 \\ 2 \end{pmatrix}$ gives us $\begin{pmatrix} 3 \\ 5 \\ 9 \end{pmatrix}$. Our

next step in decryption is to apply $(L'_2)^{-1}$ to $\begin{pmatrix} 3 \\ 5 \\ 9 \end{pmatrix}$.

$$(L'_2)^{-1}(x_1, x_2, x_3) = \begin{pmatrix} 10x_1 + x_3 \\ x_2 \\ x_1 \end{pmatrix}, \quad (L'_2)^{-1}(3, 5, 9) = \begin{pmatrix} 10(3) + (9) \\ (5) \\ (3) \end{pmatrix} = \begin{pmatrix} 6 \\ 5 \\ 3 \end{pmatrix}$$

Now we apply the inversion of F' to $\begin{pmatrix} 6 \\ 5 \\ 3 \end{pmatrix}$. First the core transformation:

$$(\beta'_1, \beta'_2, \dots, \beta'_6) = (1, 0, 0, 0, 1, 0), \quad (\alpha'_1, \alpha'_2, \dots, \alpha'_6) = (1, 0, 0, 0, 0, 1)$$

$$\begin{pmatrix} 6 \\ 5 \\ 3 \end{pmatrix} \longrightarrow \begin{pmatrix} \frac{\beta'_1(3) + \beta'_2(5) + \beta'_3}{\beta'_4(3) + \beta'_5(5) + \beta'_6} \\ \frac{\alpha'_1(3) + \alpha'_2(5) + \alpha'_3}{\alpha'_4(3) + \alpha'_5(5) + \alpha'_6} \end{pmatrix} = \begin{pmatrix} \frac{(1)(3) + (0)(5) + (0)}{(0)(3) + (1)(5) + (0)} \\ \frac{(1)(3) + (0)(5) + (0)}{(0)(3) + (0)(5) + (1)} \end{pmatrix} = \begin{pmatrix} \frac{3}{5} \\ \frac{3}{1} \end{pmatrix} = \begin{pmatrix} 5 \\ 3 \end{pmatrix}$$

For the sequential part of F' , we solve $\frac{f'_1(x_1, x_2, x_3)}{f'_4(x_1, x_2, x_3)} = 6 = \frac{x_1 x_2}{x_2}$ for unknown x_1 and known $x_2 = 5, x_3 = 3$.

$$\frac{f'_1(x_1, 5, 3)}{g'_4(x_1, 5, 3)} = \frac{x_1(5)}{(5)} = x_1 = 6$$

Thus the inversion of F' applied to $\begin{pmatrix} 6 \\ 5 \\ 3 \end{pmatrix}$ yields $\begin{pmatrix} 6 \\ 5 \\ 3 \end{pmatrix}$. Since $(L'_1)^{-1}(x_1, x_2, x_3) = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$,

$(L'_1)^{-1}(6, 5, 3) = \begin{pmatrix} 6 \\ 5 \\ 3 \end{pmatrix}$. Since this was the last step of decryption, we see that the cipher

text $\begin{pmatrix} 2 \\ 7 \\ 2 \end{pmatrix}$ decrypts to $\begin{pmatrix} 6 \\ 5 \\ 3 \end{pmatrix}$. We note from section on the small-scale example in chapter

3, it was this very same plain text $\begin{pmatrix} 6 \\ 5 \\ 3 \end{pmatrix}$ that converted to the cipher text $\begin{pmatrix} 2 \\ 7 \\ 2 \end{pmatrix}$. Therefore, we successfully used our alternate private key to decrypt, ending our small-scale example and chapter 5.

5.9 Experimental Results and Computational Complexity

The proposal for RMPKC in 1989 suggested an implementation with k of size 2^8 and $n = 5$. Our attack programmed in Magma completes cryptanalysis consistently in less than six seconds running on a personal computer with a Pentium 4 1.5 GHz processor and 256 MB of RAM. We ran several experiments at higher values of n and for larger fields k .

Our success rate in the experiments was 100% over more than 60 tests. While experimenting with smaller fields ($GF(2^8)$ or smaller), we occasionally encounter division by zero. Our alternate private key has the same chance of encountering division by zero as the original private key. However, never did our attack fail. With larger fields ($GF(2^{16})$ or larger), we never even encountered division by zero. Our attack succeeded without a glitch.

Increasing the size of the field increases the run time of the program linearly. The larger values of n cause a much greater run time and manifest the critical elements of both the public key size of the cryptosystem and the computational complexity of our cryptanalysis. Since the public key is a set of $n + 1$ quartic polynomials, its size is of order $\mathcal{O}(n^4)$.

The following table indicates the public key size, median total run time, and median

percent of total run time for each of the four steps, for various values of n as indicated. We used $|k| = 2^{16}$, which seems to be reasonable. A k of size 2^{32} would be quite reasonable as well.

n	Public Key (kBytes)	Total Run Time (sec)	Step 2 Find $\mathcal{F}$ (%)	Step 3 Define L'_1 and F' (%)	Step 4 Find $\mathcal{G}'$ (%)	Steps 5-6 Define L'_2, G' and L'_3 (%)
5	1.5	10.8	11	78	8	3
6	2.9	40.0	9	80	8	2
10	22.0	1949	15	76	8	1
14	91.8	33654	10	80	9	1

Step 2 clearly comprises the bulk of the run time. Finding of the exact denominator of F is the biggest time hog.

We wanted to count the operations required in each step, but we found that many of the time-consuming processes in our program used pre-fabricated procedures in Magma of which we haven't access to the number of operations. We estimate that step 3 is $\mathcal{O}(n^6)$, step 2 is $\mathcal{O}(n^7)$, and step 3 is $\mathcal{O}(n^9)$ regarding number of multiplications. Regardless, with the computing power available to us, we about reached the limit as to increasing values of n .

Chapter 6

Conclusions and Future Work

6.1 Summary of Accomplishments

We successfully implement a cryptanalysis of RMPKC. Our program in Magma generates alternate private keys for RMPKCs using fields of characteristic 2 and using $5 \leq n \leq 14$. The program is successful nearly every time. In rare cases, the randomization involved creates problems that are overcome with a rerun of the cryptanalysis. Nevertheless, the program leaves no doubt that RMPKC as designed is insecure.

We intend for the explanation of the mathematics used in the cryptanalysis to fully provide an understanding of how our cryptanalysis is successful. We acknowledge that in some instances we do not provide proof to explain why certain aspects of the cryptanalysis are successful. We hope that the desire to dot all of the i's and cross all of the t's might inspire other mathematicians to continue the work.

6.2 Ideas for Future Research

6.2.1 Enhancements to RMPKC

The successful cryptanalysis of RMPKC begs the question: how could we alter RMPKC to become invulnerable to our cryptanalysis? We believe that since using quartic rational expressions mark RMPKC from other MPKCs, enhancements to RMPKC should primarily focus on the 'quartic' and 'rational' aspects. Step one of our attack effectively breaks down

the rational aspect. Step two of our attack effectively breaks down the quartic aspect. Work on enhancements must thwart one or both of these initial steps.

At this point, we do not have a starting point for work to thwart step 1. RMPKC uses the same denominator for each component rational map. Perhaps some creative way could be found to add complexity via multiple denominators without adding to the degree of the resultant public key.

Efforts to thwart the application of the partial derivative attack could also be quite interesting. It may be possible to adjust the structure of F to map from k^n to k^{m+1} and G to map from k^m to k^{n+1} where $m > n$. This might make it impossible to get $R = \mathcal{F}_H$.

Another area of enhancement which might be of interest would be to improve on the core transformation, specifically to remove the weakness of the linear subspace as we have described in the details. Another possibility here would be to develop a new core transformation that would work on more than two variables.

Yet another idea we have which is not ground-breaking but rather practical is to use homogeneous polynomials throughout. Our attack in step 2 relies on converting the public key into its natural homogenous form. As we do this, one of the resultant variables used in our attack is really a dummy variable ($x_{n+1} = 1$). It is not clear to us at this point whether this would provide additional difficulty to the cryptanalysis; but it seems to be worth exploration.

Lastly enhancements to the cryptosystem could come from some of the ideas from the enhancements to MI. The plus method, the minus method, and perturbation all have potential applications in RMPKC.

6.2.2 Making the Cryptanalysis More Efficient

Our cryptanalysis is successful, but not necessarily perfectly efficient. Opportunities for efficiency improvements exist in the method used to find the denominator polynomial of F

exactly and in the method used to solve for the polynomials of the sequential parts of both F and G . Additionally we haven't reached mastery in utilizing all of the power available to us in Magma. A programmer with more expertise in Magma will likely find additional ways to improve the efficiency of the cryptanalysis.

6.2.3 Expansion of the Cryptanalysis to Cover Odd Characteristic

We focused our research on fields of characteristic 2. We acknowledge that in step 3 we rely directly on this characteristic to solve a quadratic equation. This approach fails in fields with odd characteristic, as we note in the small-scale example at the end of chapter 5. Addressing this outage is another possible investigation for an eager mathematician to take on.

6.2.4 Further Mathematical Analysis

At a few points in this presentation we mention that our experiments show a result which we are not yet ready to back up with mathematical proof. These points are obvious starting points for further research. Here is a quick list of suggestions:

1. As related to step 2 of the cryptanalysis, a statistical analysis of $\gamma(f), \dim(R)$, as well as the rare occurrences of $R \subseteq \mathcal{F}_H$ being false.
2. As related to step 3 of the cryptanalysis, exploration into the lack of uniqueness of the subspace of the span of the public key polynomials which are divided by $z_1 L'_{1,n}(x_{n-1}, x_n) + z_2 L'_{1,n}(x_{n-1}, x_n)$ where $z_1, z_2 \in k$. See the small-scale example.
3. Also as related to step 3 of the cryptanalysis, exploration into the case of the three-dimensional subspace of $\mathcal{F}$ in $\text{Span} \{ L_{1,n-1} L_{1,n}, L_{1,n-1}, L_{1,n}, 1 \}$ that $L_{1,n} + z_3$ is not in the subspace for any $z_3 \in k$. See the small-scale example.

Appendices

We include as the appendices printouts of the files used in the cryptanalysis in Magma. Consider them freeware. The entire process is performed within the environment of Magma.

This first list of files are those that the user invokes in Magma using the "load" command.

1. RunMakeNewRMPKC.txt: This program generates a public key-private key pair. The public key is stored in "NewPublicKey.txt" and the private key is stored in "NewPrivateKey.txt".
2. RunMakePlainText.txt: This program randomly chooses a plain text and stores it in "PlainText.txt".
3. RunEncrypt.txt: This program encrypts the plain text stored in "PlainText.txt" and stores the resultant cipher text in "CipherText.txt". Encryption is per the public key stored in "NewPublicKey.txt".
4. RunDecrypt.txt: This program decrypts the cipher text stored in "CipherText.txt" and stores the resultant plain text in "DecryptedCipherText.txt". Decryption is per the private key stored in "NewPrivateKey.txt". "RunDecrypt.txt" can be easily changed to decrypt per an alternate private key, such as what is stored in "AltPrivateKey.txt".
5. RunCryptanalyze.txt: This program reads the public key in "NewPublicKey.txt" and produces an alternative private key in "AltPrivateKey.txt".

The file "Parameters.txt" is used to set up the general parameters for the cryptosystem. The various programs read this file to establish the parameters to use. Therefore maintain the same parameters during an experiment. The following are parameter ranges that should work well: $p = 2$, $8 \leq \exp P \leq 32$, $5 \leq n \leq 14$. Note $n > 9$ already takes enough time to possibly try your patience.

The following is a summary of the data files that get generated by the programs:

1. NewPublicKey.txt
2. NewPrivateKey.txt
3. AltPrivateKey.txt
4. PlainText.txt
5. CipherText.txt
6. DecryptedCipherText.txt

The following files contain programs that are required by the above list of programs:

1. AttackCQRM.txt
2. AttackOverview.txt
3. ComposeIntoPK.txt
4. FinalizeGandL3.txt
5. FindAlphaBeta.txt
6. FindFspaceV6.txt
7. FindGSpace.txt
8. FindL1andF.txt

9. MakeCQRMkey.txt
10. TestCoreTransformationInversion.txt
11. TestPubKeyPrivKeyPair.txt
12. UseKeys.txt
13. FindL1andF.txt
14. UtilitiesOrderMap.txt
15. UtilitiesPrint.txt
16. UtilitiesSolver.txt
17. UtilitiesVectorPoly.txt

The actual printouts would not be included inside the text of this dissertation because of formatting issues. Therefore, we separately compiled pdf files for them and added them to this document later. The page numbering of these pages follows a separate but very logical approach. Each program file gains its own page numbering as you will plainly see.

Bibliography

Bibliography

- [1] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. *Attacking and Defending the McEliece Cryptosystem. In Proceedings of the 2nd International Workshop on Post-Quantum Cryptography - PQCrypto2008, Lecture Notes in Computer Science*, pages 31-46. Springer, 2008.
- [2] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6): 644-654, 1976.
- [3] Jintai Ding. A New Variant of the Matsumoto-Imai Cryptosystem Through Perturbation. In *Public Key Cryptosystems, PKC-2004*, volume 2947 of *Lecture Notes in Computer Science*, pages 305-318. Springer, 2004.
- [4] Jintai Ding, Jason E. Gower, and Dieter S. Schmidt. *Multivariate Public Key Cryptosystems*, volume 25 of *Advances in Information Security*. Springer, 2006.
- [5] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public Key Cryptosystems from Lattice Reduction Problems. In *CRYPTO'97: Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, pages 112-131. Springer, 1997.
- [6] L. Goubin and N.T. Courtois. Cryptanalysis of the TTM Cryptosystem. In *Advances in Cryptography – ASIACRYPT2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 44-57. Springer 2000.
- [7] L. Goubin and J. Patarin. Asymmetric Cryptography with S-Boxes. In *Proceedings of ICICS'97*, volume 1334 of *Lecture Notes in Computer Science*. Springer, 1997.
- [8] L. Goubin and J. Patarin. Trapdoor One-Way Permutations and Multivariate Polynomials. In *Proceedings of ICICS'97*, volume 1334 of *Lecture Notes in Computer Science*. Springer, 1997.
- [9] Leslie Lamport. Constructing Digital Signatures from a One-Way Function. *Technical Report SRI-CSL-98, SRI International Computer Science Laboratory*, 1979.
- [10] Tsutomu Matsumoto and Hideki Imai. Public Quadratic Polynomial-tuples for Efficient Signature-Verification and Message-Encryption. In *Advances in Cryptology – EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 419-453. Springer, 1988.
- [11] Robert J. McEliece. a Public-Key Cryptosystem Based on Algebraic Coding Theory. *Jet Propulsion Laboratory DSN Progress Report*, pages 42-44, 114-116, 1978.
- [12] Ralph C. Merkle. A Certified Digital Signature. *Advances in Cryptology*, 218-238, 1990.
- [13] T. Moh. A Public Key system With Signature and Master Key Function. *Communications in Algebra*, 27(5):2207-2222, 1999.
- [14] Phong Q. Nguyen. Cryptanalysis of the Goldreich-Goldwasser-Halevi Cryptosystem from CRYPTO'97. In *CRYPTO'99: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, pages 288-304. Springer, 1999.

- [15] Jacques Patarin. Cryptanalysis of the Matsumoto and Imai Public Key Scheme of Eurocrypt'88. In *CRYPTO'95: Proceedings of the 15th Annual International Cryptology conference on Advances in Cryptology*, pages 248-261. Springer, 1995.
- [16] Hidden Field Equations (HFE) and Isomorphisms of Polynomials (IP): Two new families of Asymmetric Algorithms. In *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 33-48. Springer, 1996.
- [17] Oil and Vinegar Signature Scheme. Dagstuhl Workshop on Cryptography, 1997.
- [18] G.V.S. Raju and Rehan Akbani. Elliptic Curve Cryptosystem and its Applications. *IEEE International Conference on Systems, Man, and Cybernetics*, 2: 1540-1543, 2003.
- [19] Oded Regev. Lattice-Based Cryptography. In *Proceedings of the 26th Annual International Cryptology Conference - Crypto2006, Lecture Notes in Computer Science*, pages 131-141. Springer, 2006.
- [20] R.L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2): 120-126, 1978.
- [21] Peter W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Rev.*, 41(2):303-332 (electronic), 1999.
- [22] S. Tsujii and A. Fujioka and Y. Hirayama. Generalization of the public key cryptosystem based on the difficulty of solving a system of non-linear equations. *ICICE Transactions (A) J72-A*, 2: 390-397, 1989.
- [23] Lih-Chung Wang, Yuh-Hua Hu, Feipei Lai, Chun-Yen Chou, and Bo-Yin Yang. Tractable rational map signature. *Public key cryptography - PKC 2005*, LNCS 3386: 244-257, 2005.
- [24] Ding-Feng Ye, Kwok-Yan Lam, and Aong-Duo Dai. Cryptanalysis of 2R Schemes, *Advances in Cryptology: CRYPTO'99*, vol. 1666 of *Lecture Notes in Computer Science*, pages 315-325. Springer, 1999.

```
//-----
//          AttackCQRM
//
// This file contains the procedure AttackCQRM, which generates
//  a private key given a CQRM public key
//
// The new private key is written to NewPrivateKeyFile.
//

AttackCQRM := procedure ( PublicKey, NewPrivateKeyFile )

//-----
// Homogenize PublicKey into HomPK and convert PublicKey into PKV

RFK := FieldOfFractions ( PRK );
HomPK := [];
PKV := [];

for i in [1..n+1] do
  HomPK [i] := PRK! ( x[n+1]^4 * RFK! Evaluate
    ( PublicKey [i], [ x[j] / x[n+1] : j in [1..n+1] ] ) );
  PKV [i] := VectorFromPoly ( PRK, n, 4, H4m, PublicKey[i], OM );
end for;

//-----
// Invoke FindFspace

temp := Cputime();

Fspace := FindFspace ( HomPK );

print "Time required to run FindFspace =", Cputime(temp), "sec";
print "";

//print Fspace;

//-----
// Invoke FindL1andF
temp := Cputime();

L1,F,FoL1,aa,bb := FindL1andF
  ( Fspace, PKV, "F" );

print "Time required to run FindL1andF =", Cputime(temp), "sec";
print "";

//-----
// Invoke FindGspace

temp := Cputime();
L3oGoL2, Gspace := FindGspace ( FoL1, HomPK );
print "Time required to run FindGspace =", Cputime(temp), "sec";
AttackCQRM.txt - 1
```

```

print "";

//-----
// Invoke FindL1andF, but this time to find L2 and G

temp := Cputime();
L2,TempGV,TempGoL2V := FindL1andF
  ( Gspace, L3oGoL2[n+1], "G" );

print "Time required to run ~FindL2andG =", Cputime(temp), "sec";
print "";

//-----
// Invoke FinalizeGandL3

temp := Cputime();
G, L3, cc, dd := FinalizeGandL3 ( TempGoL2V, TempGV, L3oGoL2 );
print "Time required to run FinalizeGandL3 =", Cputime(temp), "sec";
print "";

//-----
// Print the new Private Key to NewPrivateKeyFile

PrintMatrix ( NewPrivateKeyFile, "w", "L1inv", K, Transpose(L1^-1), n, n );
PrintMatrix ( NewPrivateKeyFile, "a", "L2inv", K, Transpose(L2^-1), n, n );
PrintMatrix ( NewPrivateKeyFile, "a", "L3inv", K, Transpose(L3^-1), n, n );

PrintVector ( NewPrivateKeyFile, "a", "a", K, aa, 6 );
PrintVector ( NewPrivateKeyFile, "a", "b", K, bb, 6 );
PrintVector ( NewPrivateKeyFile, "a", "c", K, cc, 6 );
PrintVector ( NewPrivateKeyFile, "a", "d", K, dd, 6 );

PrintList ( NewPrivateKeyFile, "a", "f", F, n+1, "PRK!" );
PrintList ( NewPrivateKeyFile, "a", "g", G, n+1, "PRK!" );

end procedure;

```

```
//-----
// Create an alternate private key from a public key

print "-----";
print "Now we create an alternate private key from a public key.";

m := n + 1;
PRK<[x]> := PolynomialRing ( K, m, "grevlex" );

DimH2m := IntegerRing()! ( NumberOfPermutations ( m+1, 2 ) / 2 );
H2m := VectorSpace ( K, DimH2m );

DimH3m := IntegerRing()! ( NumberOfPermutations ( m+2, 3 ) / 6 );
H3m := VectorSpace ( K, DimH3m );

DimH4m := IntegerRing()! ( NumberOfPermutations ( m+3, 4 ) / 24 );
H4m := VectorSpace ( K, DimH4m );

load "UtilitiesOrderMap.txt";
OM := MakeOrderMap (n,4);

load "UtilitiesVectorPoly.txt";
load "UtilitiesSolver.txt";
load "UtilitiesPrint.txt";

load "TestCoreTransformationInversion.txt";
load "ComposeIntoPK.txt";
load "FindAlphaBeta.txt";
load "FindFspaceV6.txt";
load "FindL1andF.txt";
load "FindGspace.txt";
load "FinalizeGandL3.txt";
load "AttackCQRM.txt";

load "NewPublicKey.txt";

AttackCQRM ( PublicKey, "AltPrivateKey.txt" );
```

```
//-----
//          ComposeIntoPK
//
// This file contains the function ComposeIntoPK, which calculates
//   PK given L1, F, L2, G, and L3
//-----

ComposeIntoPK := function ( L1, F, L2, G, L3 )

//-----
// Convert L1 matrix into n L1X[i] polynomials (linear)
//
// Note: multiplication corresponds to  $X * L1 = L1X$ 

L1X := [];
for i in [1..n] do
  L1X [i] := PRK! L1 [1] [i] * PRK! x[1];

  for j in [2..n] do
    L1X [i] += L1 [j] [i] * x[j];
  end for;
end for;

//-----
// Compose f with L1X, forming foL1X

L1X [n+1] := PRK!0;
foL1X := [];
for i in [1..n+1] do
  foL1X [i] := Evaluate ( F[i], [ L1X[j]: j in [1..n+1] ] );
end for;

//-----
// Compose L2 with foL1X, forming L2ofL1X
//
// Note: multiplication corresponds to  $foL1X * L2 = L2ofL1X$ 

L2ofL1X := [];
for i in [1..n] do
  L2ofL1X [i] := PRK! L2 [1] [i] * foL1X [1];
  for j in [2..n] do
    L2ofL1X [i] += L2 [j] [i] * foL1X [j];
  end for;
end for;

L2ofL1X [n+1] := foL1X [n+1];

//-----
// Homogenize G into HomG

HomG := [];
```

```

RfK := FieldOfFractions ( PRK );
for i in [1..n+1] do
  HomG [i] := PRK! ( x[n+1]^2 * RfK! Evaluate
    ( G [i], [ x[j] / x[n+1] : j in [1..n+1] ] ) );
end for;

print "HomG";
print HomG;

//-----
// Compose g with L2ofL1X, forming goL2ofL1X,
// note how the denominator function from L2ofL1X works here

goL2ofL1X := [];

for i in [1..n+1] do
  goL2ofL1X[i] := Evaluate ( HomG[i], [ L2ofL1X[j]: j in [1..n+1] ] );
end for;

//-----
// Compose L3 with goL2ofL1X, forming PublicKey
//
// Note: multiplication corresponds to goL2ofL1X * L3 = PublicKey

PublicKey:= [];
for i in [1..n] do
  PublicKey[i] := PRK! L3 [1] [i] * goL2ofL1X [1];
  for j in [2..n] do
    PublicKey[i] +:= L3 [j] [i] * goL2ofL1X [j];
  end for;
end for;

PublicKey [n+1] := goL2ofL1X [n+1];

return PublicKey;

end function;

```

```
//-----
//          FinalizeGandL3
//
// This file contains the function FinalizeGandL3, which determines
// polynomials G[1..n+1], linear transformation L3, cc, and dd
// (which are used to invert the core transformation of G)
// given TempGoL2[1..n+1] and L3oGoL2[1..n+1]
//
// It also contains
//
// which are used by FinalizeGandL3
//-----

FinalizeGandL3 := function ( TempGoL2, TempGV, L3oGoL2 )

//-----
// Find TempL3

M := KMatrixSpace ( K, n+1, DimH2m )! [ TempGoL2[j]: j in [1..n+1] ];

Temp, NullM := Solution ( M, L3oGoL2 );

TempL3 := KMatrixSpace( K, n+1, n+1 )! [ Temp [i]: i in [1..n+1] ];

//-----
// Adjust Temp3GoL5 and Temp3GV to force (n+1)st column of L6 to [0..0,1]^T

Ainv := ZeroMatrix ( K, n+1, n+1 );

M := ZeroMatrix ( K, n^2, 10 );

repeat
  for i in [n..2 by -1] do
    M := ZeroMatrix ( K, n, i-1 );
    for j in [1..n] do
      for k in [1..i-1] do
        M [j][k] := TempL3[j][k];
      end for;
    end for;

    NullM := NullSpace ( M );
    DimNullM := Dimension ( NullM );
    Temp := VectorSpace ( K, n )!0;
    for j in [1..DimNullM] do
      Temp += BasisElement ( NullM, j ) * Random ( K );
    end for;

    for j in [1..n] do
      Ainv[i][j] := Temp[j];
    end for;
  end for;
end for;
```

```

Ainv[1][1] := K!1;
Ainv[n+1][n+1] := K!1;

until Determinant ( Ainv ) ne 0;

MatB := Ainv * TempL3;

GoL2V := [];
GV := [];
for i in [1..n+1] do
  GoL2V [i] := H2m!0;
  GV[i] := H2m!0;
  for j in [1..n+1] do
    GoL2V[i] += MatB[i][j] * TempGoL2[j];
    GV[i] += MatB[i][j] * TempGV[j];
  end for;
end for;

M := KMatrixSpace ( K, n+1, DimH2m )! [ GoL2V[j]: j in [1..n+1] ];

Temp, NullM := Solution ( M, L3oGoL2 );

//print "Temp";
//print Temp;
//print "";

L3 := ZeroMatrix( K, n, n );
for i in [1..n] do
  for j in [1..n] do
    L3[i][j] := Temp[i][j];
  end for;
end for;

//print "L3";
//print L3;
//print "";

CheckL3oGoL2V := [];
for i in [1..n] do
  CheckL3oGoL2V [i] := H2m!0;
  for j in [1..n] do
    CheckL3oGoL2V [i] += L3[i][j] * GoL2V [j];
  end for;
end for;
CheckL3oGoL2V [n+1] := GoL2V [n+1];

for i in [1..n+1] do
  if CheckL3oGoL2V [i] ne L3oGoL2[i] then
    print "ERROR: CheckL3oGoL2V["i,""] <> L3oGoL2["i,""]";
  end if;
end for;

```



```
//-----
// Find cc's (alphas) and dd's (betas) to invert the core transformation

GMatrix := FindAlphaBetaPrep ( GV );
cc,dd := FindAlphaBeta ( GMatrix );

if not TestCoreInv ( GV, cc, dd ) then
  print "ERROR - the Core Transformation inversion does NOT work";
end if;

/*
//-----
// Find cc's (alphas) and dd's (betas) to invert the core transformation
//-----
// Find cc's (alphas)

M := ZeroMatrix ( K, 6, 6 );

M[1][4] := GV[4][17]; M[1][5] := GV[5][17]; M[1][6] := GV[6][17];
M[2][4] := GV[4][18]; M[2][5] := GV[5][18]; M[2][6] := GV[6][18];
M[3][1] := GV[4][17]; M[3][2] := GV[5][17]; M[3][3] := GV[6][17];
M[3][4] := GV[4][20]; M[3][5] := GV[5][20]; M[3][6] := GV[6][20];
M[4][1] := GV[4][18]; M[4][2] := GV[5][18]; M[4][3] := GV[6][18];
M[4][4] := GV[4][21]; M[4][5] := GV[5][21]; M[4][6] := GV[6][21];
M[5][1] := GV[4][20]; M[5][2] := GV[5][20]; M[5][3] := GV[6][20];
M[6][1] := GV[4][21]; M[6][2] := GV[5][21]; M[6][3] := GV[6][21];

NullM := NullSpace ( Transpose ( M ) );

if Dimension ( NullM ) ne 1 then
  print "Failure finding exact cc's - install a repeat loop";
end if;

cc := [];
NullMV := BasisElement ( NullM, 1 );
for i in [1..6] do
  cc [i] := NullMV [i];
end for;

//-----
// Find dd's (betas) - most of M is the same

M[2][4] := GV[4][20]; M[2][5] := GV[5][20]; M[2][6] := GV[6][20];
M[3][4] := GV[4][18]; M[3][5] := GV[5][18]; M[3][6] := GV[6][18];
```

Appendix File Name: FinalizeGandL3.txt

```
M[4][1] := GV[4][20]; M[4][2] := GV[5][20]; M[4][3] := GV[6][20];
```

```
M[5][1] := GV[4][18]; M[5][2] := GV[5][18]; M[5][3] := GV[6][18];
```

```
NullM := NullSpace ( Transpose ( M ) );
```

```
if Dimension ( NullM ) ne 1 then  
  print "Failure finding exact dd's - install a repeat loop";  
end if;
```

```
dd := [];  
NullMV := BasisElement ( NullM, 1 );  
for i in [1..6] do  
  dd [i] := NullMV [i];  
end for;
```

```
*/
```

```
G := [];  
for i in [1..n+1] do  
  G[i] := PolyFromVector ( PRK, n, 2, GV[i], OM );  
end for;
```

```
return G, L3, cc, dd;
```

```
end function;
```

```
//-----
//-----
//
//          FindAlphaBeta
//
//-----
//-----
//
// This file contains FindAlphaBeta and FindAlphaBetaPrep
//
// FindAlphaBeta is described below
//
// FindAlphaBetaPrep takes as input f[n-1], f[n], f[n+1]
// and prepares the matrix A for FindAlphaBeta
//
//-----
//          FindAlphaBeta
//
// This file contains the function FindAlphaBeta, which generates
// lists Alpha and Beta given matrix A, where
//
//  $f[n] = A[1][1]x[n]x[n-1] + A[1][2]x[n] + A[1][3]x[n-1] + A[1][4]$ 
//  $f[n-1] = A[2][1]x[n]x[n-1] + A[2][2]x[n] + A[2][3]x[n-1] + A[2][4]$ 
//  $f[n+1] = A[3][1]x[n]x[n-1] + A[3][2]x[n] + A[3][3]x[n-1] + A[3][4]$ 
//
//  $x[n] = \frac{Alpha[1] f[n] + Alpha[2] f[n-1] + Alpha[3] f[n+1]}{Alpha[4] f[n] + Alpha[5] f[n-1] + Alpha[6] f[n+1]}$ 
//
//  $x[n-1] = \frac{Beta[1] f[n] + Beta[2] f[n-1] + Beta[3] f[n+1]}{Beta[4] f[n] + Beta[5] f[n-1] + Beta[6] f[n+1]}$ 
//
//-----

FindAlphaBeta := function ( A )

//-----
// Form Matrix M to solve for Alpha

M := ZeroMatrix ( K, 6, 6 );

for i in [1..3] do
  for j in [1..4] do
    M[i][j+2] := A[i][j];
  end for;
end for;

for i in [4..6] do
  for j in [1..4] do
    M[i][j] := A[i-3][j];
  end for;
end for;
```

```

end for;

NullM := NullSpace ( M );
Alpha := [];
if Dimension ( NullM ) ne 0 then
    Alpha := BasisElement ( NullM, 1 );
end if;

//-----
// Alter Matrix M to solve for Beta

for i in [4..6] do
    M[i][2] := A[i-3][3];
    M[i][3] := A[i-3][2];
    M[i][4] := K!0;
    M[i][5] := A[i-3][4];
end for;

NullM := NullSpace ( M );
Beta := [];
if Dimension ( NullM ) ne 0 then
    Beta := BasisElement ( NullM, 1 );
end if;

return Alpha, Beta;

end function;

//-----
//          Function FindAlphaBetaPrep
//
// This function is specific to RMPKC (CQRM)

FindAlphaBetaPrep := function ( FV )

A := ZeroMatrix ( K, 3, 4 );
A[1][1] := FV [ n ][OM([n-1, n])];
A[1][2] := FV [ n ][OM([ n ,n+1])];
A[1][3] := FV [ n ][OM([n-1,n+1])];
A[1][4] := FV [ n ][OM([n+1,n+1])];

A[2][1] := FV [n-1][OM([n-1, n])];
A[2][2] := FV [n-1][OM([ n ,n+1])];
A[2][3] := FV [n-1][OM([n-1,n+1])];
A[2][4] := FV [n-1][OM([n+1,n+1])];

A[3][1] := FV [n+1][OM([n-1, n])];
A[3][2] := FV [n+1][OM([ n ,n+1])];
A[3][3] := FV [n+1][OM([n-1,n+1])];
A[3][4] := FV [n+1][OM([n+1,n+1])];

return A;

```

Appendix File Name: FindAlphaBeta.txt

end function;

```
//-----
//          FindFspace
//
// This file contains the function FindFspace, which generates
// a vector space (Fspace) given a CQRM public key
//

FindFspace := function
    ( HomPK )

m := n + 1;

//-----
// Define XiH2m [n+1], subspaces of H3m

VectorCoeff := H3m!0;
Vector := [];
XiH2m := [];

//for i in [1..3] do
for i in [1..n+1] do
    ctr := 1;
    for j in [1..n+1] do
        for k in [j..n+1] do
            index := OM([i,j,k]);
            VectorCoeff [ index ] := K!1;
            Vector [ ctr ] := H3m![ VectorCoeff[i]: i in [1..DimH3m] ];
            VectorCoeff [ index ] := K!0;
            ctr += 1;
        end for;
    end for;
//print sub < H3m | Vector >;
    XiH2m [i] := sub < H3m | Vector >;
end for;

//-----
// Compute the partial derivates of HomPK, forming DxPK in PRK, and
// DxPKV in H3m

DxPK:= ZeroMatrix ( PRK, n+1, n+1 );
DxPKV := [];
for i in [1..n+1] do
    DxPKV [i] := [];
    for j in [1..n+1] do
        DxPK [i] [j] := Derivative ( HomPK[i], x[j] );
        DxPKV [i][j] := VectorFromHomPoly ( PRK, n+1, 3, H3m, DxPK[i][j], OM );
    end for;
end for;

//-----
// Project DxPKV vectors onto the plane x[k]=0 ( n <= 6 ) or
// onto the plan x[k]=x[k+1]=0 ( n > 6), forming DxPKproj[1..3]
FindFspaceV6.txt - 1
```

```

ProjToZero := [];

if n le 6 then
  ProjToZero [1] := {2};
  ProjToZero [2] := {3};
  ProjToZero [3] := {1};
elif n eq 7 then
  ProjToZero [1] := {2,3};
  ProjToZero [2] := {4,5};
  ProjToZero [3] := {6,1};
else
  ProjToZero [1] := {4,2,3};
  ProjToZero [2] := {1,5,6};
  ProjToZero [3] := {7,8,9};
  ProjToZero [4] := {2,5,8};
  ProjToZero [5] := {3,6,9};
  ProjToZero [6] := {1,4,7};
  ProjToZero [7] := {4,5,9};
end if;

k := 0;
repeat
  k += 1;
  Temp := [];
  ctr := 0;
  for i in [1..n+1] do
    for j in [1..n+1] do
      if j ne k then
        ctr += 1;
        Temp [ctr] := DxPKV [i][j];
        for r in [1..n+1] do
          for s in [r..n+1] do
            for t in ProjToZero [k] do
              Temp [ctr][OM([t,r,s])] := K!0;
            end for;
          end for;
        end for;
      end if;
    end for;
  end for;
  DxPKproj := sub < H3m | Temp >;
  DxPKprojMeetX := DxPKproj meet XiH2m [k];
  DimDxPKmeetX := Dimension ( DxPKprojMeetX );

  Temp := [];
  for i in [1..DimDxPKmeetX] do
    Temp [i] := H2m!0;
    TempBasis := BasisElement ( DxPKprojMeetX, i );
    for r in [1..n+1] do
      for s in [r..n+1] do
        Temp [i][OM([r,s])] := TempBasis[OM([k,r,s])];
      end for;
    end for;
  end for;
end repeat;

```

```

    end for;
  end for;
end for;

ctr := DimDxPKmeetX;
for r in [1..n+1] do
  for t in ProjToZero[k] do
    ctr += 1;
    Temp [ctr] := H2m!0;
    Temp [ctr][OM([t,r])]:= K!1;
  end for;
end for;

// print "sub < H2m | Temp >, k =", k;
TempF := sub < H2m | Temp >;
// print "";

if k eq 1 then
  Fspace := TempF;
else
  Fspace := Fspace meet TempF;
end if;
DimF := Dimension ( Fspace );

// print "k =", k, "   Dim ( Fspace ) =", DimF;

until k eq #ProjToZero or DimF eq n + 1;

return Fspace;

end function;

```



```
//-----
//          FindGspace
//
// This file contains the function FindGspace, which generates
//   L3oGoL2 and its span, FindGspace, given PublicKey[n+1] and FoLV
//
// It also contains
//
//   which are used by FindGspace
//
//-----

FindGspace := function ( FoL, HomPK )

//-----
// Homogenize FoL into HomFoL

RFK := FieldOfFractions ( PRK );
HomFoL := [];
for i in [1..n+1] do
  HomFoL [i] := PRK! ( x[n+1]^2 * RFK! Evaluate
    ( FoL [i], [ x[j] / x[n+1] : j in [1..n+1] ] ) );
end for;

//-----
// Define matrix M
m := n + 1;
M := ZeroMatrix ( K, DimH2m, DimH4m );

for i in [1..m] do
  for j in [1..m] do
    RowM := OM([i,j]);
    Temp := HomFoL[i] * HomFoL[j];
    for k in [1..m] do
      for l in [1..m] do
        for r in [1..m] do
          for s in [1..m] do
            M [ RowM ] [ OM([k,l,r,s]) ] :=
              MonomialCoefficient ( Temp, x[k]*x[l]*x[r]*x[s] );
          end for;
        end for;
      end for;
    end for;
  end for;
end for;

//-----
// For each i (i.e. PubKey [i]), find L3oGoL2[i] s.t.
//   PubKey [i] = L3oGoL2[i] o FoL

L3oGoL2 := [];
for i in [1..m] do
```

```

B := H4m!0;
for j in [1..m] do
  for k in [j..m] do
    for r in [k..m] do
      for s in [r..m] do
        B [ OM ([j,k,r,s]) ] := MonomialCoefficient
          ( HomPK [i], x[j]*x[k]*x[r]*x[s] );
      end for;
    end for;
  end for;
end for;

L3oGoL2[i], dummy := Solution ( M, B );

// print "L3oGoL2[" , i, " ]";
// print L3oGoL2[i];
// print "";

end for;

Gspace := sub < H2m | [ L3oGoL2[i]: i in [1..n+1] ] >;

//print "Gspace";
//print Gspace;
//print "";

//-----
// Test L3oGoL2 o HomFoL; is it = PubKey

//-----
// Compose L3oGoL2 with HomFoL, forming MyPK,
// note how the denominator function from HomFoL works here

MyPK := [];
for i in [1..n+1] do
  MyPK [i] := L3oGoL2[i][1] * HomFoL [n+1]^2;

  for j in [1..n] do
    MyPK [i] += L3oGoL2[i][OM([j])] * HomFoL [j] * HomFoL [n+1];
    MyPK [i] += L3oGoL2[i][OM([j,j])] * HomFoL [j]^2;
    for k in [j+1..n] do
      MyPK [i] += L3oGoL2[i][OM([j,k])] * HomFoL[j] * HomFoL[k];
    end for;
  end for;
end for;

//-----
// Test the result

Temp := true;
for i in [1..n+1] do

```

Appendix File Name: FindGSpace.txt

```
if MyPK [i] ne HomPK [i] then
  Temp := false;
  print "ERROR in L3oGoL2 -- when composed with HomFoL, not = PubKey";
  print "i=",i;
end if;
end for;

if not Temp then
  print "FAILURE: L3oGoL2 ( HomFoL ) <> HomPK";
  print "";
end if;
```

```
//-----
```

```
return L3oGoL2, Gspace;
```

```
end function;
```

Appendix File Name: FindL1andF.txt

```
//-----
//          FindL1andF
//
// This file contains the function FindL1andF, which generates
// polynomials F[1..n+1] given Fspace and PublicKey[n+1]
//
// It also contains MakeLastWtNonZero, FillLiXj,
// FindExactFdenom, Findaabb, Testaabb, FindSeqF,
//
// which are used by FindL1andF
//-----

//-----
// Procedure MakeLastWtNonZero alters the content of two corresponding
// lists if necessary to ensure the last element has nonzero weight

MakeLastWtNonZero := procedure ( K, ~Item, ~Weight, last )

WhichToSwap := last - 1;
while Weight [ last ] eq K!0 and WhichToSwap ge 1 do
// print "MakeLastWtNonZero has encountered a zero weight";

    Temp1 := Item [ last ];
    Temp2 := Weight [ last ];

    Item [ last ] := Item [ WhichToSwap ];
    Weight [ last ] := Weight [ WhichToSwap ];

    Item [ WhichToSwap ] := Temp1;
    Weight [ WhichToSwap ] := Temp2;

    WhichToSwap -= 1;
end while;

end procedure;

//-----
// Procedure FillLiXj

FillLiXj := procedure ( ~PartLiXj, t, L, OM, H2m )

for i in [1..n] do
    for j in [1..n] do
        PartLiXj[i][OM([i,j])] := L[t][j];
    end for;
end for;

end procedure;

//-----
// Function FindExactFdenom
```

```
//
// Given: U[1..u], V[1..s], T[1..t]
// Returns: w[1..u]
//
// such that ( w[1]U[1]+...+w[u]U[u] ) ( a[1]V[1]+...+a[v]V[v] )
// in Span { T[1..t] } for some a[1..s]

FindExactFdenom := function ( K, m, U, u, V, v, T, t, OM )

DimH4m := IntegerRing()! ( NumberOfPermutations ( m+3, 4 ) / 24 );
Timer1 := Cputime();
M := ZeroMatrix ( K, u*v + t, DimH4m );

Row := 0;
for i in [1..u] do
  for j in [1..v] do
    Row += 1;
    if j lt v + i - 2 then // This makes duplicate rows all zeros
      for k in [1..m] do
        for l in [k..m] do
          for r in [1..m] do
            for s in [r..m] do
              M [Row][OM([k,l,r,s])] += U[i][OM([k,l])] * V[j][OM([r,s])];
            end for;
          end for;
        end for;
      end for;
    end if;
  end for;
end for;
for i in [1..t] do
  Row += 1;
  M [Row] := T[i];
end for;
print "Time required to build the big matrix to find exact denominator =",
  Cputime(Timer1), "sec";
print " ";

NullM := NullSpace ( M );
NullMV := BasisElement ( NullM, 1 );

//print "In function FindExactFdenom";
//print "NullM";
//print NullM;
//print "";

if NullMV eq VectorSpace ( K, u*v+t )!0 then
  print "NullMV eq VectorSpace ( K, u*v+t )!0";
  return [0];
end if;

w := [];
```

```

w[1] := K!1;

ctr := 1;
while NullMV [ctr] eq K!0 and ctr le m do
  ctr += 1;
end while;

if NullMV [ctr] eq K!0 then
  print "NullMV [ctr] eq K!0";
  return [0];
end if;

for i in [2..u] do
  w[i] := NullMV [ ctr + (i-1)*m ] / NullMV [ ctr ];
end for;

return w;

end function;

//-----
// Function Findaabb
//
// Given: Psi
// Returns: aa, bb
//
// such that aa and bb are used successfully to invert the core
// transformation created by Psi

Findaabb := function ( K, Psi )

aa := [];
bb := [];
// Find aa's (alphas)

M := ZeroMatrix ( K, 6, 6 );

M[1][3] := Psi[1][1]; M[1][4] := Psi[1][3]; M[1][5] := Psi[1][2];
M[1][6] := Psi[1][4];

M[2][3] := Psi[2][1]; M[2][4] := Psi[2][3]; M[2][5] := Psi[2][2];
M[2][6] := Psi[2][4];

M[3][3] := Psi[3][1]; M[3][4] := Psi[3][3]; M[3][5] := Psi[3][2];
M[3][6] := Psi[3][4];

M[4][1] := Psi[1][1]; M[4][2] := Psi[1][3]; M[4][3] := Psi[1][2];
M[4][4] := Psi[1][4];

M[5][1] := Psi[2][1]; M[5][2] := Psi[2][3]; M[5][3] := Psi[2][2];
M[5][4] := Psi[2][4];

```

```
M[6][1] := Psi[3][1]; M[6][2] := Psi[3][3]; M[6][3] := Psi[3][2];
M[6][4] := Psi[3][4];
```

```
NullM := NullSpace (M);
```

```
if Dimension ( NullM ) ne 1 then
  return false, aa, bb;
end if;
```

```
NullMV := BasisElement ( NullM, 1 );
for i in [1..6] do
  aa [i] := NullMV [i];
end for;
```

```
//-----
// Find bb's (betas) - most of M is the same
```

```
M[1][4] := Psi[1][2]; M[1][5] := Psi[1][3];
```

```
M[2][4] := Psi[2][2]; M[2][5] := Psi[2][3];
```

```
M[3][4] := Psi[3][2]; M[3][5] := Psi[3][3];
```

```
M[4][2] := Psi[1][2]; M[4][3] := Psi[1][3];
```

```
M[5][2] := Psi[2][2]; M[5][3] := Psi[2][3];
```

```
M[6][2] := Psi[3][2]; M[6][3] := Psi[3][3];
```

```
NullM := NullSpace (M);
```

```
if Dimension ( NullM ) ne 1 then
  return false, aa, bb;
end if;
```

```
NullMV := BasisElement ( NullM, 1 );
for i in [1..6] do
  bb [i] := NullMV [i];
end for;
```

```
return true, aa, bb;
```

```
end function;
```

```
//-----
// Testaabb tests the inversion of the core transformation of FV
// by aa and bb
```

```
Testaabb := function ( FV, aa, bb )
```

```
x1 := Random (K);
x2 := Random (K);
```

```

//print "x1 =",x1;
//print "x2 =",x2;

Colx1x2 := OM([n-1,n ]);
Colx1    := OM([n+1,n-1]);
Colx2    := OM([n+1,n ]);
ColConst := OM([n+1,n+1]);

y1 := x1*x2*FV[n-1][Colx1x2] + x1*FV[n-1][Colx1]
      + x2*FV[n-1][Colx2] + FV[n-1][ColConst];

y2 := x1*x2*FV[n][Colx1x2] + x1*FV[n][Colx1]
      + x2*FV[n][Colx2] + FV[n][ColConst];

y3 := x1*x2*FV[n+1][Colx1x2] + x1*FV[n+1][Colx1]
      + x2*FV[n+1][Colx2] + FV[n+1][ColConst];

//print "y1 =",y1;
//print "y2 =",y2;
//print "y3 =",y3;

xx1 := ( bb[1]*y1/y3 + bb[2]*y2/y3 + bb[3] )
      / ( bb[4]*y1/y3 + bb[5]*y2/y3 + bb[6] );

xx2 := ( aa[1]*y1/y3 + aa[2]*y2/y3 + aa[3] )
      / ( aa[4]*y1/y3 + aa[5]*y2/y3 + aa[6] );

//print "xx1 =",xx1;
//print "xx2 =",xx2;

if x1 ne xx1 or x2 ne xx2 then
  return false;
else
  return true;
end if;

end function;

//-----
//
//      FindSeqF
//
// The procedure finds the next iteration of FoLV[t], L[t], and FV[t]
// given FoLV, L, FV, LiXj, and t

FindSeqF := procedure ( ~M1, ~FoLV, ~L, ~FV, LiXj, H2m,
  DimH2m, t, OM )

//-----
// Build NewM1 from M

```



```

NewM1 := Submatrix ( M1, 1, 1, t, DimH2m );

//print "NewM1 - step I";
//print NewM1;
//print "";

NewM1 := VerticalJoin( NewM1,
    Submatrix ( M1, t+2, 1, (n-t-1)*n, DimH2m ) );

//print "NewM1 - step II";
//print NewM1;
//print "";

M1 := VerticalJoin ( NewM1, LiXj[t+1] );

//print "M1 - step III";
//print M1;
//print "";

/*
NumRows := IntegerRing()! ( NumberOfPermutations ( n-t, 2 ) / 2 );
NewM1 := VerticalJoin ( NewM1, Submatrix
    ( M1, (n-t-1)*n + t+1 + 1, 1, NumRows, DimH2m ) );

//print "NewM1 - step IV";
//print NewM1;
//print "";

Temp := ZeroMatrix ( K, n-t, DimH2m );
for i in [1..n-t] do
    Temp [i] := QuadL [t+1][t+i];
end for;
M1 := VerticalJoin ( NewM1, Temp );

//print "M1 - step V";
//print M1;
//print "";

*/

Temp := Submatrix
    ( M1, 1, n+2, t + (n-t)*n, DimH2m-n-1 );
//    ( M1, 1, n+2, t + (n-t)*n + NumRows + n-t, DimH2m-n-1 );

//print "Temp - Submatrix ( M1 )";
//print Temp;
//print "";

//-----
// Use NullSpace (M1) to find FoLV[t], L[t], and FV[t]

NullM1 := NullSpace ( Temp );

```

```

NullV := BasisElement ( NullM1, 1 );

//print "NullM1";
//print NullM1;
//print "";

MakeLastWtNonZero ( K, ~M1, ~NullV, t );

FoLV[t] := H2m!0;
for i in [1..t] do
  FoLV[t] += NullV [i] * M1 [i];
end for;

//print "FoLV";
//print FoLV;
//print "";

for i in [1..n] do
  L[t][i] := NullV [t+i];
end for;

//print "L";
//print L;
//print "";

FV[t] := H2m!0;
FV[t][OM([t,n])] := K!1; // by the choice of L[t]

TempVec := VectorSpace ( K, n )!0;
for i in [1..n-t-1] do
  for j in [1..n] do
    TempVec [j] := NullV [t + n*i + j];
  end for;
// print "TempVec =", TempVec;
  Soln, TempNull := Solution ( L, TempVec );

  for j in [t..n] do
    FV[t][OM([n-i,j])] += Soln [j];
  end for;

// print "TempNull";
// print TempNull;
// print "";

// print "Soln";
// print Soln;
// print "";

for j in [1..n] do
  TempVec [j] := FoLV [t][n-j+2];
end for;

```

```

// print "TempVec =", TempVec;
Soln, TempNull := Solution ( L, TempVec );

for j in [1..n] do
    FV[t][OM([j])] := Soln [j];
end for;

// print "TempNull";
// print TempNull;
// print "";

// print "Soln";
// print Soln;
// print "";

FV[t][1] := FoLV[t][1];

// print "FV";
// print FV;
// print "";

end for;

end procedure;

//-----
//-----
//
//
//      FindL1andF
//
//
//-----
//-----

FindL1andF := function
    ( Fspace, PolyZ, Flag )
m := n + 1;
H1n := VectorSpace ( K, n );

//-----
// Define TempFoL[1..n-1] directly from Fspace

TempFoL := [];
for i in [1..n-1] do
    TempFoL[i] := BasisElement ( Fspace, i );
end for;

//-----
// Define TempFoL[n..m] as the two linear components of Fspace

```

```

Temp := [];
for i in [1..m] do
  Temp [i] := H2m!0;
  Temp [i] [OM([i])] := 1;
end for;
H1inH2 := sub < H2m | Temp >;

Temp := H1inH2 meet Fspace;

TempFoL[n] := BasisElement ( Temp, 1 );
TempFoL[m] := BasisElement ( Temp, 2 );

//print "TempFoL";
//print TempFoL;
//print "";

//-----
// TempL1 = TempFoL[n], TempL2 = TempFoL[m],
// but with TempL1, TempL2 in H1n

TempL1 := H1n!0;
TempL2 := H1n!0;
for i in [1..n] do
  ColH2m := OM ([i,m]);
  TempL1 [i] := TempFoL [n] [ ColH2m ];
  TempL2 [i] := TempFoL [m] [ ColH2m ];
end for;

//-----
// Find TempFoL[n-1], the missing component of the core transformation
//
// As polynomials:
// TempL1L1 = TempL1^2
// TempL2L2 = TempL2^2
// TempL1L2 = TempL1 * TempL2
// Note: the computations are done with the vector components.
//
// We now find linear combos of TempFoL [1..n-1] in
// Span { TempL1L1, TempL2L2, TempL1L2, TempFoL[n], TempFoL[m], 1 }

TempL1L1 := H2m!0;
TempL2L2 := H2m!0;
TempL1L2 := H2m!0;
for i in [1..n] do
  Col := OM([i,i]);
  TempL1L1 [ Col ] := TempL1[i]^2;
  TempL2L2 [ Col ] := TempL2[i]^2;
  for j in [1..n] do
    TempL1L2 [ OM([i,j]) ] += TempL1[i] * TempL2[j];
  end for;
end for;

```

```

//print "TempL1L1 =", TempL1L1;
//print "TempL2L2 =", TempL2L2;
//print "TempL1L2 =", TempL1L2;
//print "";

M := ZeroMatrix ( K, n+4, DimH2m );

for i in [1..n-1] do
  M [i] := TempFoL [i];
end for;
M [n ] := TempL1L1;
M [n+1] := TempL2L2;
M [n+2] := TempL1L2;
M [n+3] := TempFoL [n];
M [n+4] := TempFoL [m];

//-----
// We must eliminate the constant term ( OM([m])=1 )

for i in [1..n+4] do
  M [i] [1] := K!0;
end for;

NullM := NullSpace (M);
//print "M";
//print M;
//print "";

if Dimension ( NullM ) ne 1 then
  print "PROBLEM WITH FINDING FoL[n-1]";
end if;

//print "Null(M) =";
//print NullSpace (M);
//print "";

BasisVNullM := BasisElement ( NullM, 1 );

//-----
// Invoke MakeLastWtNonZero to force BasisVNullM [n-1] <> 0,
//
// We must have TempFoL [n-1] not in Span { TempFoL [i]: 1<=i<=n-2 }

MakeLastWtNonZero ( K, ~TempFoL, ~BasisVNullM, n-1 );

if BasisVNullM [n-1] eq K!0 then
  print "FAILURE: BasisVNullM [1..4] = 0";
end if;

//-----
// Assign linear combos to Temp2FoL [n-1]

```

```

Temp2FoL := [];
Temp2FoL [n-1] := H2m!0;
for i in [1..n-1] do
    Temp2FoL [n-1] += BasisVNullM [i] * TempFoL [i];
end for;

//print "Temp2FoL";
//print Temp2FoL;
//print "";

//-----
// Invoke Solver1Char2 to find the right linear combinations of
// TempL1 and TempL2 to get Temp2FoL[n-1]
// in Span { TempL1*TempL2, TempFoL[n], TempFoL[m], 1 }

if BasisVNullM[n] ne K!1 then
    print "ERROR - BasisVNullM[n] <> 1";
    print "";
end if;

Result, r1, s1, r2, s2 := Solver1Char2
    ( K, expP, K!1, K!1, BasisVNullM[n+2], BasisVNullM[n+1] );

if Result ne 2 or r1 ne s2 or r2 ne s1 then
    print "NOTICE: result from Solver1Char2:";
    print "Result =", Result, " r1 =", r1, " s1 =", s1, " r2 =",
        r2, " s2 =", s2;
    print "";
end if;

L := ZeroMatrix ( K, n, n );
L[n-1] := TempL1 + r1 * TempL2;
L[n] := TempL1 + s1 * TempL2;

//print "L";
//print L;
//print "";

Temp2FoL [n] := TempFoL[n] + r1 * TempFoL[n+1];
Temp2FoL [n+1] := TempFoL[n] + s1 * TempFoL[n+1];

//print "Temp2FoL";
//print Temp2FoL;
//print "";

LiXj := [];
for i in [1..n] do
    LiXj [i] := ZeroMatrix ( K, n, DimH2m );
end for;
FillLiXj ( ~LiXj[n], n, L, OM, H2m );
FillLiXj ( ~LiXj[n-1], n-1, L, OM, H2m );

```

```

//print "LiXj";
//print LiXj;
//print "";

Temp2F := [];

Temp2F [n-1] := H2m!0;
Temp2F [n-1] [ OM([n-1,n]) ] := K!1;
Temp2F [n-1] [1] := Temp2FoL [n-1] [1];

Temp2F [n] := H2m!0;
Temp2F [n] [ OM([n]) ] := K!1;
Temp2F [n] [1] := Temp2FoL [n] [1];

Temp2F [m] := H2m!0;
Temp2F [m] [ OM([n]) ] := K!1;
Temp2F [m] [1] := Temp2FoL [m] [1];

//print "Temp2F";
//print Temp2F;
//print "";

//-----
// Find the exact denominator - F[m];

for i in [1..n-2] do
  Temp2FoL [i] := TempFoL [i];
end for;

//print "Temp2FoL";
//print Temp2FoL;
//print "";

Psi := ZeroMatrix ( K, 3, 4 );

if Flag eq "F" then
  TempList := FindExactFdenom ( K, m, [Temp2FoL[i]:i in [n-1..n+1] ],
    3, Temp2FoL, m, PolyZ, m, OM );
elif Flag eq "G" then
  M := KMatrixSpace( K, 3, DimH2m )! [ Temp2FoL[i]: i in [n-1..n+1] ];
  TempList, Dummy := Solution ( M, PolyZ );
end if;

for i in [1..3] do
  Psi[3][i] := TempList[i];
end for;

repeat
  for i in [1..2] do
    for j in [1..3] do
      Psi [i][j] := Random ( K );
    end for;
  end for;
end repeat

```

```

end for;

for i in [1..3] do
  Psi [i][4] := Temp2F [n-1] [1] * Psi [i][1]
              + Temp2F [n] [1] * Psi [i][2]
              + Temp2F [n+1] [1] * Psi [i][3];
end for;

Valid, aa, bb := Findaabb ( K, Psi );

until Valid;

//-----
// Form FV[n-1..n+1] and FoLV[n-1..n+1] using Psi's

FV := [];
FoLV := [];

for i in [1..3] do
  FV[n-2+i] := H2m!0;
  FoLV[n-2+i] := H2m!0;
  for j in [1..3] do
    FV[n-2+i] += Psi[i][j] * Temp2F[n-2+j];
    FoLV[n-2+i] += Psi[i][j] * Temp2FoL[n-2+j];
  end for;
end for;

//print "FoLV";
//print FoLV;
//print "";

//print "FV";
//print FV;
//print "";

//print "L";
//print L;
//print "";

//print "aa =",aa;
//print "bb =",bb;
//print "";

if not Testaabb ( FV, aa, bb ) then
  print "ERROR - Inversion of FV[n-1..n+1] FAILED!!!!";
  print "";
//else
// print "SUCCESSFUL inversion of FV[n-1..n+1]";
end if;

FMatrix := FindAlphaBetaPrep ( FV ); // new functions to calc aa,bb
aa,bb := FindAlphaBeta ( FMatrix );

```



```

//print "using new functions...";
//print "aa =",aa;
//print "bb =",bb;
//print "";

//-----
// Now determine the sequential components of F and the
// corresponding components of L

M1 := ZeroMatrix ( K, n-1, DimH2m );
for i in [1..n-1] do
    M1 [i] := Temp2FoL [i];
end for;
M1 := VerticalJoin ( M1, LiXj[n] );

for i in [n-2..1 by -1] do
    FindSeqF ( ~M1, ~FoLV, ~L, ~FV, LiXj, H2m, DimH2m, i, OM );
    FillLiXj ( ~LiXj[i], i, L, OM, H2m );
end for;

//print "FV";
//print FV;
//print "";

//print "L";
//print L;
//print "";

//print "FoLV";
//print FoLV;
//print "";

FoL := [];
for i in [1..n+1] do
    FoL [i] := PolyFromVector ( PRK, n, 2, FoLV[i], OM );
end for;

if sub < H2m | FoLV > ne Fspace then
    print "ERROR: Span {", Flag, "oLV } <>", Flag, "space";
//else
// print "SUCCESS: Span {", Flag, "oLV } =", Flag, "space";
end if;

Lp := [];
for i in [1..n] do
    Lp [i] := PRK!0;
    for j in [1..n] do
        Lp [i] += L[i][j] * x[j];
    end for;
end for;

```

```

Lp[n+1] := PRK!0;

//print "Lp";
//print Lp;
//print "";

Fp := [];
for i in [1..n+1] do
  Fp [i] := PolyFromVector ( PRK, n, 2, FV[i], OM );
end for;

if Flag eq "F" then
  // print "FoLV";
  // print FoLV;
  // print "";
end if;

Valid := true;
for i in [1..n+1] do
  FoLp := Evaluate ( Fp[i], Lp );
  CheckFoLp := VectorFromPoly ( PRK, n, 2, H2m, FoLp, OM );
  if CheckFoLp ne FoLV[i] then
    Valid := false;
  end if;
  // print "FoLp[" ,i, "] =", FoLp;
  // print "";
end for;
if not Valid then
  print "ERROR: FoLV <> Fp o Lp";
end if;

//print "FoLV";
//print FoLV;
//print "";

CheckFoLV := sub < H2m | FoLV >;
if CheckFoLV ne Fspace then
  print "ERROR: CheckFoLV <> Fspace";
end if;

if Flag eq "F" then
  return L, Fp, FoL, aa, bb;
else
  return L, FV, FoLV;
end if;

end function;

```

```
//-----
//          MakeCQRMkey
//
// This file contains the procedure MakeCQRMkey, which generates
//  a public key - private key pair for a CQRM cryptosystem
//
// The public key is written to PublicKeyFile.
// The private key is written to PrivateKeyFile.
//

MakeCQRMkey := procedure ( K, n, PRK, PublicKeyFile, PrivateKeyFile )

//-----
// Choose random invertible nxn matrices L1, L2, L3
//

repeat
  L1 := Random ( KMatrixSpace(K,n,n) );
until Determinant(L1) ne 0;

repeat
  L2 := Random ( KMatrixSpace(K,n,n) );
until Determinant(L2) ne 0;

repeat
  L3 := Random ( KMatrixSpace(K,n,n) );
until Determinant(L3) ne 0;

//-----
// Make L1, L2, L3 the identity matrices
//

/*
L1 := KMatrixSpace(K,n,n)!0;
L2 := KMatrixSpace(K,n,n)!0;
L3 := KMatrixSpace(K,n,n)!0;

for i in [1..n] do
  L1[i,i] := K!1;
  L2[i,i] := K!1;
  L3[i,i] := K!1;
end for;
*/

//-----
// Build the Quadratic Functions f and g, starting with the Core
// transformations and the denominator polynomials
//

f := [];
g := [];
```

```
//-----
// Choose the constants a[1]..a[6], b[1]..b[6], c[1]..c[6], d[1]..d[6]

a := Random ( VectorSpace(K,6) );
b := Random ( VectorSpace(K,6) );
c := Random ( VectorSpace(K,6) );
d := Random ( VectorSpace(K,6) );

//-----
// Make a[1],a[3],a[5],b[1],c[1],c[3],c[5],d[1] all zero

/*
a[1] := K!0;
a[3] := K!0;
a[5] := K!0;
b[1] := K!0;
c[1] := K!0;
c[3] := K!0;
c[5] := K!0;
d[1] := K!0;
*/

//-----
// Form the Denominator functions f[n+1] and g[n+1]

f [n+1] := PRK! ( a[4]*b[5] - a[5]*b[4] ) * x[n-1] * x[n] +
( a[2]*b[4] - a[1]*b[5] ) * x[n-1] +
( a[5]*b[1] - a[4]*b[2] ) * x[n] +
( a[1]*b[2] - a[2]*b[1] ) ;

g [n+1] := PRK! ( c[4]*d[5] - c[5]*d[4] ) * x[n-1] * x[n] +
( c[2]*d[4] - c[1]*d[5] ) * x[n-1] +
( c[5]*d[1] - c[4]*d[2] ) * x[n] +
( c[1]*d[2] - c[2]*d[1] ) ;

//-----
// Form the numerators of the Core Transformations

f [n-1] := PRK! ( - a[4]*b[6] + a[6]*b[4] ) * x[n-1] * x[n] +
( - a[3]*b[4] + a[1]*b[6] ) * x[n-1] +
( - a[6]*b[1] + a[4]*b[3] ) * x[n] +
( - a[1]*b[3] + a[3]*b[1] ) ;

g [n-1] := PRK! ( - c[4]*d[6] + c[6]*d[4] ) * x[n-1] * x[n] +
( - c[3]*d[4] + c[1]*d[6] ) * x[n-1] +
( - c[6]*d[1] + c[4]*d[3] ) * x[n] +
( - c[1]*d[3] + c[3]*d[1] ) ;

f [n] := PRK! ( a[5]*b[6] - a[6]*b[5] ) * x[n-1] * x[n] +
( a[3]*b[5] - a[2]*b[6] ) * x[n-1] +
( a[6]*b[2] - a[5]*b[3] ) * x[n] +
( a[2]*b[3] - a[3]*b[2] ) ;
```

```

g [n] := PRK! ( c[5]*d[6] - c[6]*d[5] ) * x[n-1] * x[n] +
              ( c[3]*d[5] - c[2]*d[6] ) * x[n-1]      +
              ( c[6]*d[2] - c[5]*d[3] ) *      x[n] +
              ( c[2]*d[3] - c[3]*d[2] )                ;

//-----
// Form the numerators of the sequential functions

for i in [1..n-2] do
  f[i] := PRK! Random (K) * x[i] + Random (K);
  g[i] := PRK! Random (K) * x[i] + Random (K);

  for j in [i+1..n] do
    f[i] += Random(K)*x[i]*x[j] + Random(K)*x[j]^2 + Random(K)*x[j];
    g[i] += Random(K)*x[i]*x[j] + Random(K)*x[j]^2 + Random(K)*x[j];

    for k in [j+1..n] do
      f[i] += Random (K) * x[j] * x[k];
      g[i] += Random (K) * x[j] * x[k];
    end for;

  end for;
end for;

//-----
// Compose ( L3 o G o L2 o F o L1 ) to form the Public Key

//-----
// Convert L1 matrix into n L1X[i] polynomials (linear)
//
// Note: multiplication corresponds to  X * L1 = L1X

L1X := [];
for i in [1..n] do
  L1X [i] := PRK! L1 [1] [i] * x[1];

  for j in [2..n] do
    L1X [i] += L1 [j] [i] * x[j];
  end for;

end for;

//print "L1X";
//print L1X;

//-----
// Compose f with L1X, forming foL1X

foL1X := [];
for i in [1..n+1] do
  foL1X [i] := Evaluate ( f[i], [ L1X[j]: j in [1..n] ] );

```

```

end for;

//print "foL1X";
//print foL1X;

//-----
// Compose L2 with foL1X, forming L2ofL1X
//
// Note: multiplication corresponds to  $foL1X * L2 = L2ofL1X$ 

L2ofL1X := [];
for i in [1..n] do
  L2ofL1X [i] := PRK! L2 [1] [i] * foL1X [1];
  for j in [2..n] do
    L2ofL1X [i] += L2 [j] [i] * foL1X [j];
  end for;
end for;

L2ofL1X [n+1] := foL1X [n+1];

//print "L2ofL1X";
//print L2ofL1X;

//-----
// Compose g with L2ofL1X, forming goL2ofL1X,
// note how the denominator function from L2ofL1X works here

goL2ofL1X := [];
for i in [1..n+1] do
  goL2ofL1X [i] := PRK!MonomialCoefficient ( g[i], 1 ) * L2ofL1X [n+1]^2;

  for j in [1..n] do
    goL2ofL1X [i] += PRK!MonomialCoefficient ( g[i], x[j] ) *
      L2ofL1X [j] * L2ofL1X [n+1];
    goL2ofL1X [i] += PRK!MonomialCoefficient ( g[i], x[j]^2 ) * L2ofL1X [j]^2;

    for k in [j+1..n] do
      goL2ofL1X [i] += PRK!MonomialCoefficient ( g[i], x[j]*x[k] ) *
        L2ofL1X [j] * L2ofL1X [k];
    end for;
  end for;
end for;

//print "goL2ofL1X";
//print goL2ofL1X;

//-----
// Compose L3 with goL2ofL1X, forming PublicKey
//
// Note: multiplication corresponds to  $goL2ofL1X * L3 = PublicKey$ 

PublicKey:= [];

```

Appendix File Name: MakeCQRMkey.txt

```
for i in [1..n] do
  PublicKey[i] := PRK! L3 [1] [i] * goL2ofol1X [1];
  for j in [2..n] do
    PublicKey[i] += L3 [j] [i] * goL2ofol1X [j];
  end for;
end for;

PublicKey [n+1] := goL2ofol1X [n+1];

//-----
// Print the Public Key to PublicKeyFile

PrintList ( PublicKeyFile, "w", "PublicKey", PublicKey, n+1, "PRK!" );

//-----
// Print the Private Key to PrivateKeyFile

PrintMatrix ( PrivateKeyFile, "w", "L1inv", K, L1^-1, n, n );
PrintMatrix ( PrivateKeyFile, "a", "L2inv", K, L2^-1, n, n );
PrintMatrix ( PrivateKeyFile, "a", "L3inv", K, L3^-1, n, n );

PrintVector ( PrivateKeyFile, "a", "a", K, a, 6 );
PrintVector ( PrivateKeyFile, "a", "b", K, b, 6 );
PrintVector ( PrivateKeyFile, "a", "c", K, c, 6 );
PrintVector ( PrivateKeyFile, "a", "d", K, d, 6 );

PrintList ( PrivateKeyFile, "a", "f", f, n+1, "PRK!" );
PrintList ( PrivateKeyFile, "a", "g", g, n+1, "PRK!" );

end procedure;
```

Appendix File Name: Parameters.txt

```
// Set up the most basic parameters
```

```
p := 2;
```

```
expP := 8;
```

```
n := 6;
```

```
q := p^expP;
```

```
K<MM> := GF(q);
```

```
//AssertAttribute( K, "PowerPrinting", true );
```

```
PRK<[x]> := PolynomialRing ( K, n, "grevlex" );
```


Appendix File Name: RunCryptanalyze.txt

```
//-----  
// CRYPTANALZE - Make an alternate private key from the public key
```

```
clear;  
load "Parameters.txt";  
load "AttackOverview.txt";
```

```
//-----  
// Test the cryptanalyzed Public Key -- Private Key pair
```

```
clear;  
print "-----";  
load "Parameters.txt";  
load "NewPublicKey.txt";  
load "AltPrivateKey.txt";  
load "TestPubKeyPrivKeyPair.txt";
```

Appendix File Name: RunDecrypt.txt

```
clear;
print "-----";
load "Parameters.txt";
load "UseKeys.txt";
load "UtilitiesPrint.txt";

// Choose whether to decrypt using "NewPrivateKey.txt" or "AltPrivateKey.txt"
//load "NewPrivateKey.txt";
load "AltPrivateKey.txt";

load "CipherText.txt";

X1 := FindPlainText
      ( Y, L1inv, L2inv, L3inv, f, g, a, b, c, d );

print "X1 (decrypted cipher text) = ", X1;
PrintVector("DecryptedCipherText.txt","w","X1",K,X1,n);

clear;
```

Appendix File Name: RunEncrypt.txt

```
clear;
print "-----";
load "Parameters.txt";
load "NewPublicKey.txt";
load "UseKeys.txt";
load "UtilitiesPrint.txt";

load "PlainText.txt";
Y := FindCipherText ( X, PublicKey );

print "Y = ", Y;
PrintVector("CipherText.txt","w","Y",K,Y,n);

clear;
```

Appendix File Name: RunMakeNewRMPKC.txt

```
print "-----";
print "-----";
print "  NEW  RMPKC  Public Key - Private Key Pair";
print "-----";
print "-----";

//-----
// To set the randomizer to get new random numbers each time.

clear;
one, two := GetSeed();
SetSeed (one + two, two + 235);
//SetSeed ( 3365754088, 2099 );
GetSeed();

clear;

//-----
// To set up the finite field and polynomial ring required for making
//  a CQRM cryptosystem.

load "Parameters.txt";

//-----
//  Make a new Public Key -- Private Key pair

load "UtilitiesPrint.txt";
load "MakeCQRMkey.txt";
MakeCQRMkey ( K, n, PRK, "NewPublicKey.txt", "NewPrivateKey.txt" );

print "K = ", K;
print "n = ", n;

//-----
//  Test the original Public Key -- Private Key pair

clear;
print "-----";
load "Parameters.txt";
load "NewPublicKey.txt";
load "NewPrivateKey.txt";
load "TestPubKeyPrivKeyPair.txt";
```

Appendix File Name: RunMakePlainText.txt

```
//-----  
// This program randomly chooses a plain text and writes it  
// to a text file  
//-----
```

```
clear;  
load "Parameters.txt";  
load "UtilitiesPrint.txt";
```

```
X := Random ( VectorSpace ( K, n ) );  
print "X = ", X;  
PrintVector("PlainText.txt", "w", "X", K, X, n);
```

```
clear;
```

```
//-----
// Testaabb tests the inversion of the core transformation of FV
// by aa and bb
```

```
TestCoreInv := function ( FV, aa, bb )
```

```
  x1 := Random (K); // x1 = x[n-1]
  x2 := Random (K); // x2 = x[ n ]
```

```
  //print "x1 =",x1;
  //print "x2 =",x2;
```

```
  Colx1x2 := OM([n-1,n ]);
  Colx1    := OM([n+1,n-1]);
  Colx2    := OM([n+1,n ]);
  ColConst := OM([n+1,n+1]);
```

```
  // y1 = y[n-1] y2 = y[n] y3 = y[n+1]
```

```
  y1 := x1*x2*FV[n-1][Colx1x2] + x1*FV[n-1][Colx1]
      + x2*FV[n-1][Colx2] + FV[n-1][ColConst];
```

```
  y2 := x1*x2*FV[n][Colx1x2] + x1*FV[n][Colx1]
      + x2*FV[n][Colx2] + FV[n][ColConst];
```

```
  y3 := x1*x2*FV[n+1][Colx1x2] + x1*FV[n+1][Colx1]
      + x2*FV[n+1][Colx2] + FV[n+1][ColConst];
```

```
  //print "y1 =",y1;
  //print "y2 =",y2;
  //print "y3 =",y3;
```

```
  xx1 := ( bb[1]*y2/y3 + bb[2]*y1/y3 + bb[3] )
        / ( bb[4]*y2/y3 + bb[5]*y1/y3 + bb[6] );
```

```
  xx2 := ( aa[1]*y2/y3 + aa[2]*y1/y3 + aa[3] )
        / ( aa[4]*y2/y3 + aa[5]*y1/y3 + aa[6] );
```

```
  //print "xx1 =",xx1;
  //print "xx2 =",xx2;
```

```
  if x1 ne xx1 or x2 ne xx2 then
    return false;
  else
    return true;
  end if;
```

```
end function;
```

```

Appendix File Name: TestPubKeyPrivKeyPair.txt
print "    Begin Test of Public & Private Key pair";

//-----
// Test the Public Key -- Private Key pair

load "UseKeys.txt";

X := Random ( VectorSpace ( K, n ) );
Y := FindCipherText ( X, PublicKey );
X1 := FindPlainText
      ( Y, L1inv, L2inv, L3inv, f, g, a, b, c, d );

if X eq X1 then
  print "SUCCESS - the public & private keys match";
else
  print "FAILURE - the public & private keys do NOT match";
  print X;
  print Y;
  print X1;
end if;

```

```
//-----
//          UseKeys
//
// This file contains the functions FindCipherText and FindPlainText.
// Each does what it says.
//
//-----
//          FindCipherText
//
// This file contains the function FindCipherText,
// which loads in the public key from PublicKeyFile and uses it
// to find the CipherText that matches the given PlainText.
//
FindCipherText:= function ( PlainText, PublicKey )

CipherTextD := Evaluate ( PublicKey[n+1], [ PlainText[j]: j in [1..n] ] );
CipherText := VectorSpace ( K, n )!0;
for i in [1..n] do
    CipherText[i]:= Evaluate ( PublicKey[i], [ PlainText[j]: j in [1..n] ] )
                    / CipherTextD;
end for;

return CipherText;

end function;

//-----
//          FindPlainText
//
// This file contains the function FindPlainText,
// which loads in the private key from PrivateKeyFile and uses it
// to find the PlainText that matches the given CipherText.
//
FindPlainText:= function
    ( CipherText, L1inv, L2inv, L3inv, f, g, a, b, c, d )

KnVectorSpace := VectorSpace ( K, n );

//-----
// Apply L3inv to CipherText to yield Z

Z := CipherText * L3inv;

//-----
// Find U=G^-1(Z), by first inverting the core transformation, and
// then solving the steps of the sequential part of g

U := KnVectorSpace!0;

U [n ] := ( c[1]*Z[n] + c[2]*Z[n-1] + c[3] ) /
```


Appendix File Name: UseKeys.txt

```
( c[4]*Z[n] + c[5]*Z[n-1] + c[6] );
U [n-1] := ( d[1]*Z[n] + d[2]*Z[n-1] + d[3] ) /
( d[4]*Z[n] + d[5]*Z[n-1] + d[6] );

// Ud is the denominator polynomial evaluated at x[n]=U[n], x[n-1]=U[n-1]

TempList := [];
for i in [1..n-2] do
  TempList [i] := PRK!0;
end for;
TempList [n-1] := PRK!U[n-1];
TempList [n] := PRK!U[n];

Ud := Evaluate ( g[n+1], TempList );

for i in [n-2..1 by -1] do
  TempList [i] := x[i];
  TempList [i+1] := PRK!U[i+1];
  Temp := Evaluate ( g[i], TempList );
  U[i] := ( Z[i]*Ud - MonomialCoefficient (Temp,1) ) /
    MonomialCoefficient (Temp, x[i]);
end for;

//-----
// Apply L2inv to U to yield W. Note Ud is not changed by L2inv

W := U * L2inv ;

//-----
// Find V = F^-1(W), by first inverting the core transformation, and
// then solving the steps of the sequential part of f

V := KnVectorSpace!0;

V [n] := ( a[1]*W[n] + a[2]*W[n-1] + a[3] ) /
( a[4]*W[n] + a[5]*W[n-1] + a[6] );
V [n-1] := ( b[1]*W[n] + b[2]*W[n-1] + b[3] ) /
( b[4]*W[n] + b[5]*W[n-1] + b[6] );

// Vd is the denominator polynomial evaluated at x[n]=V[n], x[n-1]=V[n-1]

TempList := [];
for i in [1..n-2] do
  TempList [i] := PRK!0;
end for;
TempList [n-1] := PRK!V[n-1];
TempList [n] := PRK!V[n];

Vd := Evaluate ( f[n+1], TempList );

for i in [n-2..1 by -1] do
  TempList [i] := x[i];
```

Appendix File Name: UseKeys.txt

```
TempList [i+1] := PRK!V[i+1];
Temp := Evaluate ( f[i], TempList );
V[i] := ( W[i]*Vd - MonomialCoefficient (Temp,1) ) /
        MonomialCoefficient (Temp, x[i]);
end for;

//-----
// Apply L1inv to V to yield PlainText

PlainText := V * L1inv;

return PlainText;

end function;
```

```
//-----
//      UTILITY: MakeOrderMap
//
// MakeOrderMap - creates a map (i,j,k,l)--> order of  $x[i]x[j]x[k]x[l]$ 
// using graded reverse lexicographical ordering
// (same map works for linear, quadratics, cubics, and quartics)
//
//-----
//-----
//
//      MakeOrderMap
//
//-----
// Given: v - number of variables
//      d - greatest total degree of the polynomials used
// This function creates a map from a list of integers mod v+1
// [ i[1]..i[t] ],  $t \leq d$ 
// to the order of  $x[i[1]]x[i[2]] \dots x[i[t]]$ ,
//
// Order: the location in the list of coordinates
//
// if v = 3 and deg = 2 then
//
// Vector [ a b c d e f g h j k ]
//
// corresponds to
//
//  $a + b*x[3] + c*x[2] + d*x[1] + e*x[3]^2 + f*x[3]*x[2] + g*x[3]*x[1]$ 
//  $+ h*x[2]^2 + j*x[2]*x[1] + k*x[1]^2$ 
//
// Reverse lexicographical ordering is used and the monomials are ordered
// from the least to the greatest

function MakeOrderMap ( v, deg )

if not deg in {2,3,4} or not v in {2..20} then
    return 0;
end if;

Z2 := ResidueClassRing(2);
PRZ2<[y]> := PolynomialRing ( Z2, v, "grevlex" );

//-----
// Make a d-degree polynomial of PRZ2 with every possible term

A := PRZ2!1;

for i in [1..v] do
    for j in [i..v] do
```

```

if deg gt 2 then for k in [j..v] do

    if deg gt 3 then for l in [k..v] do
        A += y[i]*y[j]*y[k]*y[l];
    end for; end if;

    A += y[i]*y[j]*y[k];
end for; end if;

    A += y[i]*y[j];
end for;
    A += y[i];
end for;

//-----
// Make a list of the terms of this polynomial

MonosOfA := Terms ( A );

//-----
// Define the map relations

Zv := ResidueClassRing (v+1);

MapRelations := [];
MaxIndex := Index ( MonosOfA, PRZ2!1 );

x := [];
for i in [1..v] do
    x[i] := PRZ2!y[i];
end for;
x[v+1] := PRZ2!1;

case deg:

when 4:
Zd := CartesianPower ( Zv, deg );
counter := 0;
for i in [1..v+1] do
    for j in [1..v+1] do
        for k in [1..v+1] do
            for l in [1..v+1] do
                counter += 1;
                MapRelations [counter] := < < Zv!i, Zv!j, Zv!k, Zv!l >,
                    MaxIndex - Index ( MonosOfA, x[i]*x[j]*x[k]*x[l] ) + 1 >;
            end for;
        end for;
    end for;
end for;
OrderMap1 := map < Zd -> Integers() | MapRelations >;

when 3:

```

```

Zd := CartesianPower ( Zv, deg );
counter := 0;
for i in [1..v+1] do
  for j in [1..v+1] do
    for k in [1..v+1] do
      counter += 1;
      MapRelations [counter] := < < Zv!i, Zv!j, Zv!k >,
        MaxIndex - Index ( MonosOfA, x[i]*x[j]*x[k] ) + 1 >;
    end for;
  end for;
end for;
OrderMap1 := map < Zd -> Integers() | MapRelations >;

when 2:
Zd := CartesianPower ( Zv, deg );
counter := 0;
for i in [1..v+1] do
  for j in [1..v+1] do
    counter += 1;
    MapRelations [counter] := < < Zv!i, Zv!j >,
      MaxIndex - Index ( MonosOfA, x[i]*x[j] ) + 1 >;
  end for;
end for;
OrderMap1 := map < Zd -> Integers() | MapRelations >;

end case;

OrderMapF := function ( a )
  b := [];
  for i in [1..deg] do
    if IsDefined ( a, i ) then
      b[i] := a[i];
    else
      b[i] := v+1;
    end if;
  end for;
  return OrderMap1(< Zv!(b[i]): i in [1..deg] >);
end function;

return OrderMapF;

end function;

```

```

//-----
//          PRINT UTILITY PROCEDURES
//
// PrintMatrix - prints Matrix to an output file
// PrintVector - prints vector to an output file
// PrintList   - prints a list to an output file
//
//-----
//          PrintMatrix
//
// This procedure writes a Matrix to a text file in a way that it can be
// read by Magma directly, i.e.
// MatrixName := KMatrixSpace(~)![ 0, 1, ..., 0 ];
//
// WriteType = "a" - append to existing file
//           = "w" - start the file over

PrintMatrix := procedure
    ( FileName, WriteType, MatrixName, K, MatrixToPrint, rows, columns )

WriteMatrix := Open ( FileName, WriteType );

if WriteType eq "a" then
    Puts( WriteMatrix, "" );
end if;

Put( WriteMatrix, Sprint( MatrixName ) cat " := KMatrixSpace( K, "
    cat Sprint( rows ) cat ", " cat Sprint ( columns ) cat " )![ "
    cat Sprint( MatrixToPrint[1][1] ) );

for j in [2..columns-1] do
    Put( WriteMatrix, ", " cat Sprint( MatrixToPrint[1][j] ) );
end for;

if rows eq 1 then
    Puts( WriteMatrix, ", " cat Sprint( MatrixToPrint[1][columns] ) cat " ];";
else
    Puts( WriteMatrix, ", " cat Sprint( MatrixToPrint[1][columns] ) );

for i in [2..rows-1] do
    for j in [1..columns-1] do
        Put( WriteMatrix, ", " cat Sprint( MatrixToPrint[i][j] ) );
    end for;
    Puts( WriteMatrix, ", " cat Sprint( MatrixToPrint[i][columns] ) );
end for;

for j in [1..columns-1] do
    Put( WriteMatrix, ", " cat Sprint( MatrixToPrint[rows][j] ) );
end for;
Puts( WriteMatrix, ", " cat Sprint( MatrixToPrint[rows][columns] ) cat " ];";

end if;

```

delete WriteMatrix;

end procedure;

```
//-----
//          PrintVector
//
// This procedure writes a Vector to a text file in a way that it can be
// read by Magma directly, i.e.
// VectorName := KVectorSpace(~)[ 0, 1, ..., 0 ];
//
// WriteType = "a" - append to existing file
//           = "w" - start the file over

PrintVector := procedure
    ( FileName, WriteType, VectorName, K, VectorToPrint, columns )

WriteVector := Open ( FileName, WriteType );

if WriteType eq "a" then
    Puts( WriteVector, "" );
end if;

Put( WriteVector, Sprint( VectorName ) cat " := VectorSpace( K, "
    cat Sprint ( columns ) cat " )![ " cat Sprint( VectorToPrint[1] ) );

for j in [2..columns-1] do
    Put( WriteVector, ", " cat Sprint( VectorToPrint[j] ) );
end for;

Puts( WriteVector, ", " cat Sprint( VectorToPrint[columns] ) cat " ];" );

delete WriteVector;

end procedure;
```

```
//-----
//          PrintList
//
// This procedure writes a List to a text file in a way that it can be
// read by Magma directly, i.e.
// ListName := [];
// ListName [1] := 1;
//
// WriteType = "a" - append to existing file
//           = "w" - start the file over

PrintList := procedure
    ( FileName, WriteType, ListName, ListToPrint, GrtestIndex, ElementPrefix )
UtilitiesPrint.txt - 2
```

Appendix File Name: UtilitiesPrint.txt

```
WriteList := Open ( FileName, WriteType );

if WriteType eq "a" then
  Puts( WriteList, "" );
end if;

Puts( WriteList, Sprint( ListName ) cat " := [];" );

for j in [1..GrtestIndex] do
  if IsDefined ( ListToPrint, j ) then
    Puts( WriteList, Sprint( ListName ) cat "[" cat Sprint (j)
      cat "]" := " cat ElementPrefix cat Sprint( ListToPrint[j] ) cat ";" );
  end if;
end for;

delete WriteList;

end procedure;
```



```

//-----
//          UTILITIES - Solver
//
// Solver1Char2 - solves ( az + by = c, zy = d ) for z,y
// Solver2Char2 - solves ( z^2 + az + b = 0 ) for z
// SolverMultinomialQuad - solves a1,...,at, b1, ...,bt
//                       given a1b1, a1b2+a2b1, ..., atbt
//
//-----
//
//      Solver2Char2
//
//-----
// solves ( y^2 + ay + b = 0 ) for y
//
// Returns NumSolns, y1, y2

Solver2Char2 := function ( K, expP, a, b )

//-----
// Define PRKzz[zz], a version of K with expP "variables" from GF(2),
//   created as a quotient ring of PRKz by setting zz[i]^2 = zz[i].

PRKz<[zz]> := PolynomialRing ( K, expP ); // lexicographical ordering

BasisTempIdeal := [];
for i in [1..expP] do
    BasisTempIdeal [i] := zz[i]^2 + zz[i];
end for;

TempIdeal := ideal < PRKz | BasisTempIdeal >;
PRKzz := PRKz / TempIdeal;

KasVS, IsoKtoVS := VectorSpace ( K, GF(2) );
IsoKtoVSInv := IsoKtoVS^-1;

r := PRKzz!0;
for i in [1..expP] do
    r += PRKzz!zz[i] * PRKzz!MM^(i-1);
end for;

//print "r =", r;
//print "r^2 =", r^2;
//print "a * r =", a * r;

Sum := r^2 + PRKzz!a * r;

//print "Sum =", Sum;

Coeffzz := [];
for i in [1..expP] do
    Coeffzz [i] := K!MonomialCoefficient ( Sum, zz[i] );

```

```

end for;

//print "Coeffzz";
//print Coeffzz;
//print "";

CoeffzzV := [];
for i in [1..expP] do
  CoeffzzV [i] := IsoKtoVS ( Coeffzz [i] );
end for;

//print "CoeffzzV";
//print CoeffzzV;
//print "";

M := ZeroMatrix ( GF(p), expP, expP );
for i in [1..expP] do
  M [i] := CoeffzzV [i];
end for;

bV := IsoKtoVS ( b );

//print "bV =", bV;

if Dimension ( sub <KasVS | bV> meet RowSpace(M) ) eq 0 then
  NumSolns := 0; y1 := K!0; y2 := K!0;
else
  yV, NullM := Solution ( M, bV );
  //print "NullM";
  //print NullM;
  //print "";
  y1 := IsoKtoVSInv ( yV );

  if Dimension ( NullM ) eq 1 then
    y2 := y1 + IsoKtoVSInv ( BasisElement ( NullM, 1 ) );
    NumSolns := 2;
  else
    y2 := y1;
    NumSolns := 1;
  end if;
end if;

return NumSolns, y1, y2;

end function; // end of Solver2Char2

//-----
//
//   Solver1Char2
//
//-----
// solves ( az + by = c, zy = d ) for z,y

```

```
//
// returns Result, z1,y1,z2,y2
//
// If there are no solutions, Result=0, z1=y1=z2=y2=0
//
// If there are many (>2) sol'ns, Result=3, z1=y1=z2=y2=0
//
// If there is one unique solution, Result=1, z2=y2=0
//
// If there are two solutions, Result=2,(z1,y1),(z2,y2) are sol'ns

Solver1Char2 := function ( K, expP, a, b, c, d )
z1 := K!0; y1 := K!0; z2 := K!0; y2 := K!0;

if d ne K!0 then
  if a * b ne K!0 then
    NumSolns,z1,z2 := Solver2Char2 ( K, expP, c / a, b * d / a );
    if NumSolns gt 0 then
      y1 := d / z1;
      if NumSolns eq 2 then
        y2 := d / z2;
        Result := 2;
      else
        z2 := K!0;
        Result := 1;
      end if;
    else
      Result := 0;
      print "No solutions from Solver2Char2";
    end if;
  elif a ne K!0 then
    if c ne K!0 then
      z1 := c / a;
      y1 := d / z1;
      Result := 1;
    else
      Result := 3;
    end if;
  elif b ne K!0 then
    if c ne K!0 then
      y1 := c / b;
      z1 := d / y1;
      Result := 1;
    else
      Result := 3;
    end if;
  elif c ne K!0 then
    Result := 0;
  else
    Result := 3;
  end if;
elif a * b * c ne K!0 then
```

```

z1 := c / a;
y2 := c / b;
Result := 2;
elif a * b ne K!0 then
  Result := 1;
elif c ne K!0 then
  Result := 1;
  if a ne K!0 then
    z1 := c / a;
  elif b ne K!0 then
    y1 := c / b;
  else
    Result := 0;
  end if;
else
  Result := 3;
end if;

return Result, z1, y1, z2, y2;

end function; // end of Solver1Char2

```

```

/*          // This function needs help to work = an order map
//-----
//
//   SolverMultinomialQuad
//
//-----
// solves a1,...,at, b1, ...,bt given a1b1, a1b2+a2b1, ... atbt
//
// returns NumSolns, a, b
//
// NumSolns = number of linearly independent solutions found
//
// (a[1],b[1])...(a[NumSolns],b[NumSolns]) are linearly independent
//   solution sets
//
// where (a[j][1],...,a[j][t],b[j][1],...,b[j][t]) is a solution

SolverMultinomialQuad := function ( K, t, Q )

TM := MakeLexOrderMap ( t, 2 );
Zd := ResidueClassRing ( t );
Zd2 := CartesianPower ( Zd, 2 );

print "Q =",Q;

```

```

NumSolns := 0;
a := [];
b := [];

i := 0;
repeat
    i += 1;
until Q[TM(< Zd!i, Zd!i >)] ne K!0
    or i gt t;

if i lt t then
    print "Good here, i =", i;
    ai := K!1;
    bi := Q[TM(< Zd!i, Zd!i >)];

    j := i;
    repeat
        j += 1;
    until ( Q[TM (< Zd!j, Zd!j >)] ne K!0 and
        Q[TM(< Zd!i, Zd!j >)] ne K!0 ) or j gt t;

    if j le t then
        print "Good here, too; j =", j;
        NumSolns, TestA1, TestB1, TestA2, TestB2:= Solver1Char2
            ( K, expP, bi, ai, Q[TM(< Zd!i, Zd!j >)],
              Q[TM(< Zd!j, Zd!j >)] );
        print "NumSolns =", NumSolns, "from Solver1Char2";

        if NumSolns ge 1 then // Since no parameters of Solver1char2 are 0,
            a[1] := []; // we won't get Numsolns = 3
            b[1] := [];
            a[1][i] := ai;
            b[1][i] := bi;
            a[1][j] := TestA1;
            b[1][j] := TestB1;
            if NumSolns eq 2 then
                a[2] := [];
                b[2] := [];
                a[2][i] := a[1][i];
                b[2][i] := b[1][i];
                a[2][j] := TestA2;
                b[2][j] := TestB2;
            end if;

            TempValid := [];
            for r in [1..NumSolns] do
                k := 0;
                TempValid [r]:= true;
                repeat
                    repeat
                        k += 1;
                    until k ne i and k ne j;

```

Appendix File Name: UtilitiesSolver.txt

```

a[r][k] := ( a [r][i] * Q[TM(< Zd!k, Zd!j >)]
            + a [r][j] * Q[TM(< Zd!k, Zd!i >)] )
            / Q[TM (< Zd!i, Zd!j >)];
b[r][k] := ( b [r][i] * Q[TM(< Zd!k, Zd!j >)]
            + b [r][j] * Q[TM(< Zd!k, Zd!i >)] )
            / Q[TM (< Zd!i, Zd!j >)];
TempValid [r] := a[r][k]* b[r][k] eq Q[TM(<Zd!k,Zd!k>)];
if not TempValid [r] then
    print "no longer is our solution valid";
    print "k =", k;
    print "r =", r;
end if;
until k eq t or not TempValid [r];

k := 1;
l := 2;

while TempValid [r] and k lt t do
    if k ne i and k ne j and l ne i and l ne j then
        TempValid [r] := a[r][k]*b[r][l] + a[r][l]*b[r][k]
                        eq Q[TM(<Zd!k,Zd!l>)];
    end if;

    if l eq t then
        k += 1;
        l := k + 1;
    else
        l += 1;
    end if;
end while;
end for;

if NumSolns eq 2 then
    if not TempValid [2] then
        a[2] := [];
        b[2] := [];
        NumSolns := 1;
    elif not TempValid [1] then
        a[1] := a[2];
        b[1] := b[2];
        Numsolns := 1;
        TempValid [1]:= TempValid [2];
    end if;
end if;
if NumSolns eq 1 and not TempValid [1] then
    a := [];
    b := [];
    NumSolns := 0;
end if;
end if;
else
    print "Found Q(i,i)<>0; but for all j, Q(i,j)*Q(j,j)=0";
    UtilitiesSolver.txt - 6

```

```

    // This could be addressed
    end if;
elif i eq t then
    print "Q(t,t)<>0; but for all other i, Q(i,i)=0";
else
    print "Never found i s.t. Q(i,i)<>0";
end if;

*/

/*
if Valid then // must use NumSolns here
    i := 1;
    repeat
        if a[i]*b[i] ne Q[TM(< Zd!i, Zd!i >)] then
            Valid := false;
        elif i lt t then
            j := i + 1;
            repeat
                if a[i]*b[j]+a[j]*b[i] ne Q[TM(< Zd!i, Zd!j >)] then
                    Valid := false;
                else
                    j += 1;
                    if j gt t then
                        i += 1;
                        end if;
                    end if;
                until j gt t or not Valid;
            end if;
        until i eq t or not Valid;
    end if;
*/

/*
return NumSolns, a, b;

end function;

*/

```

```
//-----
//      UTILITIES: Vector - Polynomial Conversions
//
// VectorFromPoly   - converts a polynomial into a vector
// VectorFromHomPoly - converts a homogeneous polynomial into a vector
// PolyFromVector   - converts a vector into a polynomial
// HomPolyFromVector - converts a vector into a homogeneous polynomial
//
//-----
//      VectorFromPoly
//-----
// Given:
// PR   - polynomial ring
// NumVar - number of variables in the polynomial ring
// Deg   - the maximum total degree of the polynomial
// VS    - the vector space of the returned vector
// Poly  - the polynomial to be converted
// OM    - the map used to convert the term to the component
//
// This function returns a vector matching the polynomial

VectorFromPoly := function ( PR, NumVar, Deg, VS, Poly, OM )

Vector := VS!0;
for i in [1..NumVar] do
  if Deg gt 1 then for j in [1..NumVar] do
    if Deg gt 2 then for k in [1..NumVar] do
      if Deg gt 3 then for l in [1..NumVar] do
        Vector [OM([i,j,k,l])] :=
          MonomialCoefficient ( Poly, x[i]*x[j]*x[k]*x[l] );
      end for; end if;
    end for; end if;
    Vector [OM([i,j,k])] :=
      MonomialCoefficient ( Poly, x[i]*x[j]*x[k] );
  end for; end if;
  Vector [OM([i,j])] := MonomialCoefficient ( Poly, x[i]*x[j] );
end for; end if;
  Vector [OM([i])] := MonomialCoefficient ( Poly, x[i] );
end for;
Vector [1] := MonomialCoefficient ( Poly, 1 );

return Vector;

end function;

//-----
//
//      VectorFromHomPoly
//-----
//-----
```



```
// Given:
// PR   - polynomial ring
// NumVar - number of variables in the polynomial ring
// Deg   - the maximum total degree of the polynomial
// VS    - the vector space of the returned vector
// Poly  - the polynomial to be converted
// OM    - the map used to convert the term to the component
//
// This function returns a vector matching the polynomial

VectorFromHomPoly := function ( PR, NumVar, Deg, VS, Poly, OM )

Vector := VS!0;
for i in [1..NumVar] do
  if Deg gt 1 then for j in [1..NumVar] do
    if Deg gt 2 then for k in [1..NumVar] do
      if Deg gt 3 then for l in [1..NumVar] do
        Vector [OM([i,j,k,l])] :=
          MonomialCoefficient ( Poly, x[i]*x[j]*x[k]*x[l] );
      end for; else
        Vector [OM([i,j,k])] :=
          MonomialCoefficient ( Poly, x[i]*x[j]*x[k] );
      end if;
    end for; else
      Vector [OM([i,j])] := MonomialCoefficient ( Poly, x[i]*x[j] );
    end if;
  end for; else
    Vector [OM([i])] := MonomialCoefficient ( Poly, x[i] );
  end if;
end for;

return Vector;

end function;

//-----
//
//   PolyFromVector
//
//-----
// Given:
// PR   - polynomial ring
// NumVar - number of variables in the polynomial ring
// Deg   - the maximum total degree of the polynomial
// V     - the vector to be converted
// OM    - the map used to convert the term to the component
//
// This function returns a polynomial matching the vector

PolyFromVector := function ( PR, NumVar, Deg, V, OM )

Poly := PR!V[1];
```

```

for i in [1..NumVar] do
  if Deg gt 1 then for j in [i..NumVar] do
    if Deg gt 2 then for k in [j..NumVar] do
      if Deg gt 3 then for l in [k..NumVar] do
        Poly += V[OM([i,j,k,l]])*x[i]*x[j]*x[k]*x[l];
      end for; end if;
      Poly += V[OM([i,j,k]])*x[i]*x[j]*x[k];
    end for; end if;
    Poly += V[OM([i,j]])*x[i]*x[j];
  end for; end if;
  Poly += V[OM([i]])*x[i];
end for;

return Poly;

end function;

//-----
//
//   HomPolyFromVector
//
//-----
// Given:
// PR   - polynomial ring
// NumVar - number of variables in the polynomial ring
// Deg   - the maximum total degree of the polynomial
// V     - the vector to be converted
// OM    - the map used to convert the term to the component
//
// This function returns a polynomial matching the vector

HomPolyFromVector := function ( PR, NumVar, Deg, V, OM )

Poly := PR!0;
for i in [1..NumVar] do
  if Deg gt 1 then for j in [i..NumVar] do
    if Deg gt 2 then for k in [j..NumVar] do
      if Deg gt 3 then for l in [k..NumVar] do
        Poly += V[OM([i,j,k,l]])*x[i]*x[j]*x[k]*x[l];
      end for; else
        Poly += V[OM([i,j,k]])*x[i]*x[j]*x[k];
      end if;
    end for; else
      Poly += V[OM([i,j]])*x[i]*x[j];
    end if;
  end for; else
    Poly += V[OM([i]])*x[i];
  end if;
end for;

return Poly;

```

end function;

Appendix File Name: UtilitiesVectorPoly.txt