NTNU – Trondheim
Norwegian University of
Science and Technology

Classification of Keys in MQQ-SIG

Håkon Jacobsen

Master of Science in Communication Technology
Submission date: June 2012
Supervisor: Danilo Gligoroski, ITEM
Co-supervisor: Simona Samardjiska, ITEM

Norwegian University of Science and Technology
Department of Telematics

Thesis title: Classification of keys in MQQ-SIG
Student's name: Håkon Jacobsen
Problem description:

Recently, a new multivariate signature scheme MQQ-SIG has been proposed by Gligoroski et al. In its construction it uses a non-associative structure called Multivariate Quadratic Quasigroups (MQQ's), making its design different from the main classes of multivariate public-key cryptosystems. This system shows excellent properties in terms of speed, so it is very attractive for future cryptanalysis. As in any public-key cryptography scheme, it is very important for the designers to analyze the behavior of the system for different types of keys, i.e. to distinguish the best possible pool of keys in terms of security. The most important part of the private key in MQQ-SIG, is a particular MQQ chosen from a pool defined by the construction in use. On average, these MQQ's show very promising results, however experiments have displayed different levels of deviation from the average, due to the underlying key.

This research will consist of an in depth experimental and theoretical analysis of the keys used, and their classification. The final objective is to determine the properties of the MQQ's that lead to solid or weak keys.

Assignment given: 23.01.2012
Supervisor: Danilo Gligoroski
Co-supervisor: Simona Samardjiska

Abstract

The security of almost all public-key cryptography is based on some computationally hard problem. Most prominent are the problems of factoring integers into primes, and computing discrete logarithms in finite groups. However, in the last two decades, several new public-key schemes have emerged that base their security on some completely different problems. One promising proposal, is to base the security of public-key cryptography on the difficulty of solving large systems of multivariate quadratic polynomial equations. A major challenge in designing these public-key systems, is to embed an efficient trapdoor into the set of equations. Recently, a novel approach towards this problem was suggested by Gligoroski et al. [GMK08b], using the concept of quasigroup string transformations. In this thesis, we describe a methodology for identifying strong and weak keys in the newly introduced multivariate public-key signature scheme MQQ-SIG, which is based on the idea of quasigroup string transformations.

We have conducted a large number of experiments, based on Gröbner basis attacks, in order to classify the various parameters that determine the keys in MQQ-SIG. Our findings show that there are big differences in the importance of these parameters. The methodology consists of a classification of different parameters in the scheme, together with an introduction of concrete criteria on which keys to avoid, and which to use. Additionally, we identified an unnecessary requirement in the original specification, requiring the quasigroups to fulfill a certain condition. Removing this restriction can potentially speed up the key generation process by a large factor. Having all this, we propose a new enhanced key generation algorithm for MQQ-SIG, which will generate stronger keys and be more efficient than the original key generation method.

Sammendrag

Sikkerheten til nesten all offentlig nøkkel-kryptografi er basert på et vanskelig beregnbarhetsproblem. Mest velkjent er problemene med å faktorisere heltall i sine primtallsfaktorer, og å beregne diskrete logaritmer i endelige sykliske grupper. I de to siste tiårene, har det imidlertid dukket opp en rekke andre offentlig nøkkel-systemer, som baserer sin sikkerhet på helt andre type problemer. Et lovende forslag, er å basere sikkerheten på vanskeligheten av å løse store likningsett av flervariable polynomlikninger. En stor utfordring ved å designe slike offentlig nøkkel-systemer, er å integrere en effektiv "falluke" (trapdoor) inn i likningssettet. En ny tilnærming til dette problemet ble nylig foreslått av Gligoroski m.f. [GMK08b], hvor de benytter konseptet om kvasigruppe-strengtransformasjoner (quasigroup string transformations). I denne masteroppgaven beskriver vi en metodikk for å identifisere sterke og svake nøkler i det nylig foreslåtte multivariable offentlig nøkkel-signatursystemet MQQ-SIG, som er basert på denne idéen.

Vi har gjennomført et stort antall eksperimenter, basert på Gröbner basis angrep, for å klassifisere de ulike parametrene som bestemmer nøklene i MQQ-SIG. Våre funn viser at det er store forskjeller i viktigheten av disse parametrene. Metodikken består i en klassifisering av de forskjellige parametrene i systemet, i tillegg til en innføring av konkrete kriterier for hvilke nøkler som bør velges. Videre, har vi identifisert et unødvendig krav i den originale spesifikasjonen, som krevde at kvasigruppene måtte oppfylle et bestemt kriterie. Ved å fjerne denne betingelsen, kan nøkkel-genererings-algoritmen potensielt øke ytelsen med en stor faktor. Basert på alt dette, foreslår vi en ny og forbedret nøkkel-genereringsalgoritme for MQQ-SIG, som vil generere sterkere nøkler og være mer effektiv enn den originale nøkkel-genereringsalgoritmen.

Acknowledgments

First of all, I would like to thank Simona Samardjiska for her dedicated support and encouragement throughout the course of this thesis work. Patiently, she has answered my numerous questions, and helped me on as I got stuck on some misguided idea (which happened more than once). Her comments and suggestions have made an invaluable contribution to the result of this thesis. For this I am truly grateful.

Last, but not least, I want to thank Professor Danilo Gligoroski for his continuous belief in my work. His energy and high spirit have been a huge motivating factor for me, spurring me on towards a better result.

Contents

List of Figures

List of Tables

List of Algorithms

List of Symbols and Abbreviations

A_1, A_2	Boolean matrices used in MQQ-SIG.
a, b, c	Coefficients of polynomials.
B	Boolean matrix used in MQQ-SIG.
$\mathbf{c}$	Constant vector used in MQQ-SIG.
D	Degree of regularity.
$\mathbb{E}$	Extension field of a base field $\mathbb{F}$.
$\mathbb{F}$	Generic finite field, usually of q elements.
$\mathbb{F}_q, \mathrm{GF}(q)$	Finite (Galois) field of q elements.
$\mathbb{F}_q^*$	The multiplicative group of a finite field $\mathbb{F}_q$.
m	Number of equations.
$\mathcal{MQ}$-scheme	Public-key cryptography system based on multivariate quadratic polynomials.
MQQ	Multivariate quadratic quasigroup.
MQQ-SIG	Digital signature scheme based on multivariate quadratic quasigroups.

n	Number of variables.
$O(), o()$	Standard big-O and small-o notation.
$\mathcal{P}$	The public key in $\mathcal{MQ}$-schemes.
$\mathcal{P}'$	The central map.
Q	A quasigroup $(Q, *)$, usually finite.
$\mathrm{Quad}_{d-k}\mathrm{Lin}_k$	MQQ of 2^d elements, with $d-k$ quadratic Boolean functions.
$\mathrm{Quad}^s_{d-k}\mathrm{Lin}^s_k$	Strict MQQ.
r	Equations removed from the public key.
$\mathcal{S}, \mathcal{S}'$	The initial linear or affine maps in $\mathcal{MQ}$-schemes.
S	Matrix of linear maps $\mathcal{S}$, $\mathcal{S}'$.
σ_0^0, σ_0^1	Permutations in MQQ-SIG, determining S.
$\mathcal{T}$	The final linear map in $\mathcal{MQ}$-schemes.
$U(\mathbf{x})$	Matrix of linear expression in $\mathbf{x}$ used in MQQ-SIG.
$\mathbf{v}$	Constant part of affine map $\mathcal{S}'$.
$X = X_1 \cdots X_k$	String of quasigroup elements, input to $\mathcal{P}'$.
$\mathbf{x} = (x_1, \ldots, x_n)$	Either a signature or plaintext block.
$x \in_U \mathcal{X}$	Element from $\mathcal{X}$, chosen uniformly at random.
$Y = Y_1 \cdots Y_k$	String of quasigroup elements, output from $\mathcal{P}'$.
$\mathbf{y} = (y_1, \ldots, y_n)$	Either a digest or ciphertext block.
$\mathbb{Z}_p^*$	The multiplicative group modulo p.

CHAPTER 1

Introduction

Cryptography was originally a means for the military to protect their secret communication, but has today become an indispensable tool for securing and safeguarding all aspects of life on the Internet. While encryption is still one of its fundamental application areas, modern cryptography offers a host of other essential functionalities which do not necessarily involve secrecy. Examples include digital signatures for proving authenticity; hash algorithms to prevent tampering; certificates for establishing trust; password authentication for user identification; zero-knowledge proofs to prove the possession of secrets without revealing them; digital coins that can be used just like normal currency; pseudo-random number generators for enabling fair online games; secure online voting; and so on. This list is far from exhaustive, but works to illustrate the amazing potential of cryptography. In some sense, cryptography has gone from being a means to an end, to itself being the enabler of new business and application possibilities.

With an ever increasing demand for the security capabilities provided by cryptography, it has also become increasingly important that the building blocks that underlies it are fundamentally sound. Hence, the modern field of cryptography has been tasked with this very challenge; trying to put the building of secure systems on a solid foundation. In this process, numerous schemes have been proposed and almost as many have been broken. The cryptosystems we use today, are those few

that have been able to remain unscathed from years of dedicated cryptanalytic efforts at breaking it.

However, for a cryptosystem to be useful, security is not enough; almost as important is efficiency. Numerous cryptographic schemes exist that are just as secure as those we use today, but are simply too inefficient or unpractical to be employed in any real-life system. This illustrates the delicate nature of designing cryptosystems. On the one hand they must be extremely secure, while on the other, must work efficiently under real-life conditions and constraints. Thus, the act of designing secure and efficient cryptographic schemes strikes a fine balance between highly theoretical analysis, and concrete engineering considerations.

Recently, a promising new digital signature scheme, called MQQ-SIG, was proposed by Gligoroski et al. [GØJ+11]. This system is several orders of magnitude faster than the most widely used public-key signature schemes today, and 5 to 20 times faster than other comparable systems. However, little cryptanalysis has been conducted on it, so its real security is still largely unknown. This thesis provides the first steps towards a better understanding of the characteristics of MQQ-SIG. In particular, our new contribution is an extensive experimental study of the parameters of the scheme, and how they affect the security of the system. Our end goal is to determine a classification of the parameters that lead to either strong or weak keys. In general, we are most interested being able to avoid the weak keys, since we assume that the average case provides sufficient security.

The thesis is structured as follows: Chapter 2 to Chapter 4 provides the necessary background to understand the workings of MQQ-SIG, where in particular Chapter 4 provides the full specification of the scheme; Chapter 5 contains our new research and experiments. The real meat of the chapter is four concrete recommendations on how the various parameters should be chosen in MQQ-SIG in order to avoid generating weak keys; in Chapter 6 we implement a modified key generation algorithm based on these recommendations, and compare its performance against the original; Chapter 7 concludes the thesis.

1.1 Public-key cryptography

The notion of public-key cryptography was first publicly proposed in 1976 by Diffie and Hellman in their seminal paper [DH76]. Before this, all secure communication between two parties required a pre-established key to be shared prior to their exchange of private messages. Unfortunately, there is an inherent *asymmetry* in the way our modern communication works, which makes the requirement of mutual knowledge of the same key a very bad fit. For instance, with e-commerce and online-banking, there is a clear need to be able to establish secure connections with people you most probably never have met (and who might even be on the other side of the earth from you). With the advent of public-key cryptography this problem could actually be solved.

The basic idea of public-key cryptography is the following: let Alice and Bob denote two parties wanting to communicate securely*. Prior to their communication, both Alice and Bob will have generated a *key pair*, (K_A, k_a) and (K_B, k_b), respectively. K_A is Alice's *public key*, and she should make it publicly available, for instance by posting it on her homepage. Bob would do the same with his public key. k_a and k_b are their *private keys*, and are kept secret from everyone else. Now, when Bob wants to send a message m to Alice, he gets the public K_A from her website and encrypts the message with it using the public-key encryption function E. That is, he computes $c = E(K_A, m)$, and sends c to Alice. When Alice receives c, she decrypts it using the decryption function D and her private key k_a. That is, she finds m as $m = D(k_a, c)$.

The crucial point of the above scheme is that the private/public key-pairs, together with the functions E and D, should have a special property; namely that the only person capable of decrypting a message encrypted under the public key, should be the owner of the corresponding private key. Because of this property, public-key cryptography is also called *asymmetric cryptography*.

Fundamental to the construction of public-key cryptography is the concept of a *one-way trapdoor function*. A one-way trapdoor function is a mathematical function, with the property of being easy to compute in one direction, but virtually impossible to invert without given some additional information about the "trapdoor".

---

*Note that Alice and Bob represent generic entities; not necessarily human beings. To the contrary; they will most likely *not* be humans, but rather computer programs.

More specifically, a function $f_t(x)\colon \mathcal{D} \mapsto \mathcal{R}$ is said to be one-way, if it is easy to evaluate for all $x \in \mathcal{D}$, but hard to invert for all images $f_t(x)$ in $\mathcal{R}$. It is called a one-way trapdoor function, if knowledge of some extra information t makes it easy to compute $x \in \mathcal{D}$ satisfying $y = f_t(x)$ for all $y \in \mathcal{R}$.

To fully formalize what it means for a function to be "easy" or "hard" to compute, one usually states the precise definitions using notions taken from complexity theory. For instance, a function might be regarded as easy to compute if the time to evaluate it for any $x \in \mathcal{D}$ is polynomially bounded in some parameter k. Conversely, it is hard if the time to compute it is not bounded by any polynomial in k. Of course, the exact definitions are much more involved than what we have presented here, often involving probabilistic arguments as well. However, for the purposes of this thesis, it will suffice to confine ourselves to the intuitive understanding of what it means for a function to be hard or easy to compute.

When we in the following discuss some concrete candidate[†] one-way trapdoor functions, we will be content with an informal treatment. Nevertheless, it should be implicitly understood that every statement about a problem's "hardness" could be qualified according to a formal complexity theoretic definition.

1.2 The Discrete Logarithm problem

In their paper [DH76], Diffie and Hellman suggested a cryptographic scheme which allowed two parties to agree on a shared secret over a public channel. This famous algorithm, now simply known as the Diffie-Hellman protocol, is based on the one-way properties of the modular exponentiation function $x \mapsto x^k \pmod p$, and is presented in Protocol 1.1. It is straightforward to verify that both Alice and Bob will compute the same shared key. For let k_{alice} and k_{bob} represent the values Alice and Bob derives in Step 3 and 4 in Protocol 1.1, respectively; by the rules of exponentiation we have

$$k_{alice} = g_{bob}^a \pmod p \equiv (g^b)^a \equiv g^{ab} \equiv (g^a)^b \equiv g_{alice}^b \pmod p = k_{bob}.$$

Hence, Alice and Bob have effectively established a common secret over a public channel.

[†]We say *candidate*, because the (provable) existence of one-way functions is still an open problem. In particular, their existence would also imply that $\mathcal{P} \neq \mathcal{NP}$ (see [Gol01, Chap. 2]).

Protocol 1.1 Diffie-Hellman Key Agreement

Input: p a large prime, g a generator for $\mathbb{F}_p^*$.
Output: An element $k \in \mathbb{F}_p^*$ shared between Alice and Bob.

1. Alice picks $a \in_U \mathbb{F}_p^*$. Then she computes

$$g_{alice} \leftarrow g^a \pmod{p}$$

   and sends g_{alice} to Bob.

2. Bob picks $b \in_U \mathbb{F}_p^*$. Then he computes

$$g_{bob} \leftarrow g^b \pmod{p}$$

   and sends g_{bob} to Alice.

3. Alice computes $k \leftarrow g_{bob}^a \pmod{p}$.

4. Bob computes $k \leftarrow g_{alice}^b \pmod{p}$.

The security of the Diffie-Hellman protocol relies on the assumption that calculating *discrete logarithms* is hard in finite cyclic groups. This problem can be stated more precisely as follows.

Definition 1.1. Discrete Logarithm Problem (DL Problem)
Input: A multiplicative group $(G, \cdot)$, an element $g \in G$ having order n, and an element $h \in \langle g \rangle$.
Output: The unique integer $0 \leq a \leq n-1$, such that

$$g^a = h.$$

The integer a is called the *discrete logarithm* of h, and denoted $\log_g h$.

The above definition formulates the problem in the most general way using a cyclic subgroup in some generic group G. In our statement of the Diffie-Hellman protocol we have used the special case of cyclic groups in finite fields $\mathbb{F}_p^*$, but it works similarly in any other finite cyclic group.

Now, let $g_{alice} = g^a$ and $g_{bob} = g^b$ be the messages sent by Alice and Bob, respectively. If the discrete problem was easy, then an adversary eavesdropping on the communication could calculate $a = \log_g g_{alice}$ and $b = \log_g g_{bob}$ from g_{alice} and g_{bob}, and thereby obtain $k = g^{ab}$. This is exactly the key Alice and Bob will agree on. Clearly, for the Diffie-Hellman protocol to be secure the DL problem needs to be hard. That is, the exponentiation function modulo the group order needs to have the one-way property. It is generally believed that this is the case.

Unfortunately, the hardness of the DL problem is not sufficient to guarantee the security of the Diffie-Hellman protocol, because an adversary is not strictly required to find the two private integers a and b in order to obtain the secret key k. To see this, assume the attacker have managed to obtain the two transmitted values $g_{alice} = g^a$ and $g_{bob} = g^b$. Then, it is conceivable that there might exist some way to obtain the value $k = g^{ab}$ directly, without first calculating the discrete logarithms. This latter problem is known as the *Computational Diffie-Hellman problem* (CDH).

Definition 1.2. Computational Diffie-Hellman Problem

Input: A multiplicative group $(G, \cdot)$, an element $g \in G$ having order n, and two elements $h_1, h_2 \in \langle g \rangle$.

Output: An element $h_3 \in \langle g \rangle$ such that $\log_g h_3 \equiv \log_g h_1 \times \log_g h_2 \pmod{n}$. That is, the CDH problem asks the following question: given g_1^h and g_2^h, find $g^{h_1 h_2}$.

The CDH problem is the true problem an adversary faces when attacking the Diffie-Hellman protocol. Hence for the protocol to be secure we need to additionally assume that the CDH problem is hard as well. Then how does the CDH problem relate to the DL problem?

It is trivial to see that if one could solve the DL problem then the CDH problem would be easy as well. Given two values h_1 and h_2, simply calculate their discrete logarithms and take their product. In other words, a CDH problem instance is reducible to an equivalent DL problem instance. However, the converse is not true: given a solver for the CDH problem, it is not known how one can turn it into a solver for the DL problem. The DL and CDH problems can therefore not be said to be equivalent given our current understanding. In particular, assuming that the CDH problem is hard is a *stronger* assumption than assuming that the DL problem is hard. Consequently, it cannot be ruled out that there might exist an algorithm

that solves the CDH problem, and hence breaks the Diffie-Hellman protocol, but does not also solve the DL problem. Nevertheless, most algorithms for breaking Diffie-Hellman (and other systems based on the CDH problem, such as ElGamal) focus on the more general problem of finding discrete logarithms. Therefore, it is interesting to examine the complexity of the algorithms that compute discrete logarithms.

Methods for finding discrete logarithms have been studied extensively in the last few decades. Some algorithms are *generic*, in the sense that they do not depend on any particular property of the group, but can be applied to any kind of group. Examples of these types of algorithms include the Pollard Rho algorithm [Pol78], the Pohlig-Hellman algorithm [PH78] and Shank's Baby-Step-Giant-Step algorithm [Sha71]. Other algorithms are more specialized and depends on certain properties of the group in order to be applicable. Examples of these algorithms are the Index Calculus methods [Adl79] and the Number Field Sieves [Gor93], which only works in groups from $\mathbb{Z}_p^*$ and $\mathbb{F}_{p^n}^*$.

The generic algorithms have a complexity of $O(\sqrt{n})$, where n is the order of the (sub)group; a result shown to be optimal by Shoup [Sho97b]. In the special cases of groups (and subgroups) in $\mathbb{Z}_p^*$ and $\mathbb{F}_{p^n}^*$, the number field sieve methods are actually sub-exponential [Odl99], having an expected run time of

$$O(e^{(1+o(1))\sqrt{\ln p \ln \ln p}}).$$

As a consequence of these results, p is usually required to be around 1024 bits in order for the discrete logarithm problem to be considered secure today.

1.3 The factorization problem

Two years after the publication of Diffie and Hellman's paper and the invention of public-key cryptography, Rivest, Shamir and Adleman published their now famous RSA encryption scheme [RSA78]. RSA is probably the most widely used public-key cryptosystems in the world today. The RSA function is defined as $x \mapsto x^e \pmod{n}$, where n is a composite integer and $e < n$. Initially it might look similar to the one-way function used in the Diffie-Hellman protocol, but the crucial difference is that n is not prime. This property makes it possible to embed a trapdoor into the function,

enabling efficient inversion. Therefore, the RSA function can readily be turned into an encryption scheme. We present the RSA cryptosystem in Algorithm 1.2.

Algorithm 1.2 The RSA Cryptosystem

Key Setup
User Alice sets up her public keys and parameters as follows:

1. generate two large random prime numbers p and q;
2. compute $n = pq$;
3. compute $\phi(n) = (p-1)(q-1)$;
4. choose a random integer $e < \phi(n)$ such that $\gcd(e, \phi(n)) = 1$. Find the unique integer d such that
$$ed \equiv 1 \pmod{\phi(n)}.$$

Her public key is (e, n), and her private key is (d, n).

Encryption
For the sender Bob to transmit a message $m < n$ to Alice, he creates the ciphertext as follows
$$c \leftarrow m^e \pmod{n}.$$

Decryption
To decrypt a received ciphertext c, Alice computes
$$m \leftarrow c^d \pmod{n}.$$

It is easy to verify that encryption and decryption are inverse operations in RSA. Let c be a ciphertext created as per Algorithm 1.2 of the plaintext message m, that is, $c \equiv m^e \pmod{n}$. Since
$$ed \equiv 1 \pmod{\phi(n)},$$
there exists an integer $k \geq 1$ such that
$$ed = k \cdot \phi(n) + 1.$$
Now, when Alice decrypts c she obtains
$$c^d \equiv (m^e)^d \equiv m^{ed} \equiv m^{k\cdot\phi(n)+1} \equiv (m^{\phi(n)})^k m \equiv m \pmod{n},$$

where the last congruence holds due to Euler's theorem[‡]. Thus, RSA is a valid encryption scheme.

The difficulty of breaking the RSA cryptosystem, or more precisely the RSA function, is that of finding the e-th root of c modulo n. This is the *RSA problem.*

Definition 1.3. RSA Problem

Input: A composite integer $n = pq$, where p and q are primes, an integer e such that $\gcd(e, \phi(n)) = 1$, and a ciphertext $c \in \mathbb{Z}_n^*$.

Output: The unique integer $m \in \mathbb{Z}_n^*$ satisfying $m^e \equiv c \pmod{n}$.

When the trapdoor d is known the RSA problem can be solved efficiently, but without it the problem is believed to be hard. Hence the security of the RSA cryptosystem is conditioned on the assumption that the RSA problem is hard. However, the difficulty of the RSA problem is further dependent on the difficulty of yet another problem, namely the *integer factorization problem.*

Definition 1.4. Factorization Problem

Input: An odd composite integer n, with at least two distinct prime factors.

Output: A prime p such that $p|n$.

If one could efficiently factor composite integers then it would be trivial to break the RSA system. To see this, assume an attacker has obtained the factorization of n. Then he can easily compute $\phi(n)$, from which the trapdoor d can be found solving the equation $ed \equiv 1 \pmod{\phi(n)}$. So factoring the public modulus yields a straightforward attack on the RSA problem. The reverse implication however, is not known. That is, given a solver for the RSA problem, can it be turned into a solver for the factorization problem? This is still an open problem.

Interestingly, unlike the case of the DL and CDH problems, there are many known attacks on RSA that does not involve factoring the modulus. Typically, these attacks target the private or the public key directly. Examples include Wiener's attack on low private exponents [Wie90] and Coppersmith's attack on low public exponents [Cop97]. These results, among others, suggest that the RSA problem most likely is not equivalent to the factorization problem, but rather is strictly easier. Boneh provides further remarks on this, and description of other attacks, in

[‡] $a^{\phi(n)} \equiv 1 \pmod{n}$ for all $a \in \mathbb{Z}$, where $\gcd(a, n) = 1$, and ϕ is the Euler function.

his survey paper on the RSA problem [Bon99]. Although these alternative avenues of attack on the RSA problem exits, they can usually be prevented by a suitable selection of parameters and being careful about the implementation. For a properly implemented RSA system, factoring seems to be the only viable approach. Hence, we will briefly mention the best methods for factoring integers.

Algorithms for factoring large integers have been studied for a very long time, with active research still ongoing today. Several advanced methods have been proposed, including the Quadratic Sieve algorithm, the Elliptic Curve method and the General Number Field Sieve (see Lenstra's survey [Len00] for a description of these, and many other algorithms for factoring integers). All three methods have a sub-exponential running time. Currently the fastest method is an algorithm based on the General Number Field Sieve having an expected run time complexity of

$$O(e^{(1.923+o(1))(\ln n)^{1/3}(\ln \ln n)^{2/3}}).$$

As a result of the efficiency of these sophisticated algorithms, together with the continuous increase in computer power, today's recommended size for the modulus n is 1024 bits or more.

1.4 Alternative one-way trapdoor functions

While cryptosystems based on the hardness of factorization and discrete logarithms are ubiquitous today, there exists other candidate one-way (trapdoor) functions. Patarin and Goubin [PG97] describe a list of such candidates. However, given that we already have two candidate one way functions that accounts for almost all of today's public-key cryptography, it is appropriate to ask if there is any incentive to study alternative methods. To answer this question, there are two key factors to consider: 1. security - a new proposal should of course be no less secure then today's schemes, but is there a compelling reason to switch from the well studied problems of factorization and discrete logarithms? 2. efficiency - public-key cryptography is undeniably very slow and use rather large keys when compared to symmetric cryptography schemes. Could alternative methods mitigate these problems?

We saw in the two previous sections that the best algorithms known for solving either the discrete logarithm problem or the factorization problem, are subexpo-

nential in the size of its key parameter. This is already undesirable for a secure cryptosystem, since ideally it should have been exponential. Not to say that today's public-key cryptography is believed to be unsafe, but it does raise the question of whether fully polynomial algorithms might exist for these problems. Could sub-exponential run times be the fundamental limit on how fast we can possibly hope to compute discrete logarithms and factor integers? Thirty years of research on these problems have not lead to anything better, but it is difficult to say if this is due to some inherent property of these problems, or if we simply have not found the right method yet.

Surprisingly, in the imagined future of quantum computers, cryptosystems based on the hardness of either factorization or discrete logarithms are broken. This is due to the startling discovery made by Shor, who found that quantum computers could solve these problems very efficiently [Sho97a]. While it is highly uncertain whether anyone will actually be able to build a large quantum computer, Shor's algorithm has spurred research into cryptographic schemes that can withstand even computers with these capabilities. This concept is known as *post-quantum cryptography*.

Several systems have been suggested as candidates for post-quantum cryptography. Common to them all is that they are not based on factoring or discrete logarithms. Examples include McEliece's cryptosystem based on error correcting codes [McE78]; Merkle's hash based digital signature system [Mer89]; a whole family of schemes based on lattice problems [Ajt96, HHGPW10]; and finally multivariate quadratic public-key systems, which is the topic of this thesis (see [BBD08] for an introduction to all the schemes mentioned above, including a discussion on their complexity and key sizes). For all these systems, or rather their underlying one-way function, no attack better than exponential is known, even in the presence of quantum computers, which makes them good candidates to base a cryptographic scheme upon.

This leads us to the second point made above, namely efficiency. Many of the candidates for post-quantum cryptography use very large keys, compared to conventional public-key cryptosystems. As we noted in the two previous sections, systems based on either discrete logarithms or factoring currently use key sizes of around a few thousand bits (or less in the case of elliptic curve cryptography), whereas the candidates mentioned above can have key sizes ranging in the millions

of bits. Besides the obvious increase in storage requirements, large keys makes the system more complex and can potentially slow it down.

On the other hand, current public-key cryptography is based on modular arithmetic, which involves rather expensive operations on most CPU's. Cryptosystems such as RSA, DSA[§] and ECDSA[¶], are thousands of times slower than their symmetric key counterparts. Additionally, most of the algorithms for computing modular arithmetic are inherently sequential, so public-key cryptography based on discrete logarithms and factorization can not take advantage of multiple processors to speed up their computations. At the same time, there seems to be a trend among CPU manufacturers to prefer adding more cores to their chips, rather than increasing their raw processing power. Combined, these two facts indicate that classical public-key cryptography will most likely continue to be orders of magnitude slower than symmetric crypto.

Many of the new cryptosystems mentioned above are already faster then current public-key schemes, despite their larger key sizes. By taking advantage of parallelism in their designs, they can potentially be made even faster, even rivaling the speeds of symmetric cryptography. Thus, the main motivation for studying alternatives to the discrete logarithm and the factorization problem, is the search for more efficient public-key cryptography schemes.

[§]DSA - the Digital Signature Algorithm. Standardized by the American National Institute of Standards and Technology (NIST) and based on the discrete logarithm problem.
[¶]ECDSA - Elliptic Curve DSA. The elliptic curve variant of DSA.

CHAPTER 2

Multivariate Quadratic Cryptography

In this chapter we will study an alternative one-way trapdoor function to the ones used in Diffie-Hellman and RSA; one that is not based on the hardness of computing discrete logarithms nor factoring integers. We will be looking at systems of m multivariate quadratic ($\mathcal{MQ}$) polynomial equations, in n variables, over a finite field $\mathbb{F}$. The one-way property of a set of multivariate quadratic equations comes from the fact that it is allegedly very difficult to solve over finite fields. So, while it is easy to evaluate all the polynomials at some given values from the finite field, it is generally very hard to find its pre-image. The naive brute-force approach, simply trying all possible combinations of values from $\mathbb{F}$ until the pre-image is found, will be exponential in the number of variables, more precisely $O(|\mathbb{F}|^n)$. More sophisticated algorithms exist, but no one achieves run times better than exponential. Therefore, systems of multivariate quadratic equations over finite fields are good candidates for a one-way function.

The challenging part however, is how one can embed a trapdoor into the set of equations, as to make it useful in public-key cryptography. Matsumoto and Imai are usually credited for constructing the first public-key cryptosystem based on multivariate quadratic equations, when they in 1988 proposed their

C^* scheme [MI88]. The second $\mathcal{MQ}$-system, called the Stepwise Triangular Scheme (STS), was introduced in 1993 by Shamir [Sha93], and later a variant of it, called TTM, was suggested by Moh [Moh99]. A third $\mathcal{MQ}$-scheme was proposed by Patarin in 1996, and is called Hidden Field Equations (HFE) [Pat96]. Finally, a fourth scheme, known as Unbalanced Oil and Vinegar (UOV), was suggested in 1999 by Kipnis, Patarin and Goubin [KPG99]. Unfortunately, most of these systems have been broken in their basic forms.

Until recently, the suggested schemes above have represented the only known classes on how to create a $\mathcal{MQ}$-trapdoor function. Then, in 2008, Gligoroski et al. came up with a completely new approach, called MQQ [GMK08b, GMK08a], which is based on the theory of quasigroups (we will consider quasigroups further in Chapter 3). Their latest proposal is a signature variant of the MQQ scheme called MQQ-SIG, and will be the main topic of this thesis (Chapter 4). In this chapter we will look at the basics of $\mathcal{MQ}$-based cryptography in general. We will mostly limit our discussion to the overall design ideas, but will give a brief introduction to a few of the earlier schemes mentioned above.

2.1 The $\mathcal{MQ}$-problem

Multivariate quadratic cryptography is based on the problem of solving systems of multivariate quadratic equations over a finite field. Let $\mathcal{P}$ be a system of m multivariate polynomial equations in n variables over a finite field $\mathbb{F}$

$$\mathcal{P} := \begin{cases} y_1 = p_1(x_1, \ldots, x_n) \\ y_2 = p_2(x_1, \ldots, x_n) \\ \quad \vdots \\ y_m = p_m(x_1, \ldots, x_n), \end{cases} \tag{2.1}$$

where $y_i \in \mathbb{F}$. Each p_k is a polynomial of degree two over $\mathbb{F}$ of the form:

$$p_k(x_1, \ldots, x_n) = \sum_{1 \le i \le j \le n} a_{ij}^{(k)} x_i x_j + \sum_{i=1}^{n} b_i^{(k)} x_i + c^{(k)},$$

where $a_{ij}^{(k)}, b_i^{(k)}, c^{(k)} \in \mathbb{F}$ and $1 \le k \le m$. The coefficients a_k, b_k and c_k are called the *quadratic*, *linear* and *constant* parts of the polynomial p_k, respectively. A solution of

this system, is a vector $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{F}^n$ so that all the polynomial equations are satisfied simultaneously, that is, $p_k(\mathbf{x}) = y_k$ for all $1 \leq k \leq m$. If $\mathbf{y} \in \mathbb{F}^m$ is the vector holding all the values y_k, that is, $\mathbf{y} = (y_1, \ldots, y_m)$, then we will write this more simply as $\mathcal{P}(\mathbf{x}) = \mathbf{y}$.

The *$\mathcal{MQ}$-problem* then, asks to find a solution $\mathbf{x} \in \mathbb{F}^n$ to the system in Equation 2.1. We denote the class of $\mathcal{MQ}$-problems, containing problems defined by polynomial vectors such as $\mathcal{P}$, with $\mathcal{MQ}(\mathbb{F}^n, \mathbb{F}^m)$. We formally state the $\mathcal{MQ}$-problem as follows:

Definition 2.1. **$\mathcal{MQ}$-Problem**
Input: An instance $\mathcal{P} \in \mathcal{MQ}(\mathbb{F}^n, \mathbb{F}^m)$.
Output: $\mathbf{x} \in \mathbb{F}^n$ such that $\mathcal{P}(\mathbf{x}) = \mathbf{y}$.

The $\mathcal{MQ}$-problem is known to be hard, specifically, it can be shown to be $\mathcal{NP}$-complete. A proof of this fact in the general case of $\mathbb{F}$ being any finite field, can be found in [PG97]. However, the special case of $\mathbb{F} = \mathrm{GF}(2)$ was already shown to be $\mathcal{NP}$-complete by Fraenkel [FY79]. While $\mathcal{NP}$-hardness is a good baseline on which to asses the difficulty of a problem, it is not enough by itself to guarantee a secure cryptosystem.

Firstly, $\mathcal{NP}$-hardness tells us only something about a problem's worst-case behavior, but says nothing about its average-case characteristic. The existence of very efficient SAT-solvers for practical problem instances (see the survey by Gomes et al. [GKSS08]), illustrates the discrepancy between a $\mathcal{NP}$-hard problems alleged difficulty and its actual difficulty on real-world problems.

Secondly, a real-life implementation of cryptosystem based on a hard problem might not have a reduction to the problem itself. That is, the problem of breaking the cryptosystem might be strictly easier than solving the underlying problem itself. There are many examples of this, one famous example being the system proposed by Merkle and Hellman [MH78], which was based on the knapsack problem. The Knapsack problem is $\mathcal{NP}$-complete, but the cryptosystem itself was completely broken by Shamir [Sha82].

Lastly, it might be difficult to embed an efficient trapdoor into the problem. Without this, real life implementations will not be practical.

As can be understood from the above, it is a long way from the discovery of a problems with the potential of being a one-way trapdoor-function, to actually

building a secure and efficient cryptosystem from it. The $\mathcal{MQ}$-problem then, is no exception.

2.2 Public-key cryptography based on the $\mathcal{MQ}$-problem

We will now describe the general construction of public-key systems based on the $\mathcal{MQ}$-problem. In a $\mathcal{MQ}$-system the public key is the set of multivariate equations $\mathcal{P}$. The goal is to embed a trapdoor into this set of polynomial equations, so that it will be possible to invert it efficiently. Since the generic $\mathcal{MQ}$-problem is $\mathcal{NP}$-hard, the system of equations cannot simply be generated at random, because then there would be no way to include a trapdoor into it. Thus, $\mathcal{MQ}$-systems are designed with an internal algebraic structure in order to make inversion viable. It is hoped that systems created in such a way will still behave sufficiently like a set of randomly generated equations.

Key generation

The public key $\mathcal{P}$ is usually constructed as the composition of three functions $\mathcal{S}$, $\mathcal{P}'$ and $\mathcal{T}$, that is $\mathcal{P} = \mathcal{T} \circ \mathcal{P}' \circ \mathcal{S}$. The triple $(\mathcal{S}, \mathcal{P}', \mathcal{T})$ is the private key. The two functions $\mathcal{S} : \mathbb{F}^n \to \mathbb{F}^n$ and $\mathcal{T} : \mathbb{F}^m \to \mathbb{F}^m$ are linear or affine transformations, while the central map $\mathcal{P}' : \mathbb{F}^n \to \mathbb{F}^m$ is a quadratic function. Since linear and affine transformations are easily invertible, the challenge is to make the central map $\mathcal{P}'$ invertible. The various $\mathcal{MQ}$-schemes can be classified roughly on how they construct $\mathcal{P}'$. The purpose of the transformations $\mathcal{S}, \mathcal{T}$, is to hide the algebraic structure present in the central map $\mathcal{P}'$. A graphical representation of the general design is shown in Fig 2.1.

Encryption and signature verification

Encryption and signature verification is performed in approximately the same manner for all $\mathcal{MQ}$-schemes. Let $\mathbf{x} \in \mathbb{F}^n$ be a plaintext message to be encrypted, or a signature to be verified. To either encrypt or verify $\mathbf{x}$, we simply evaluate

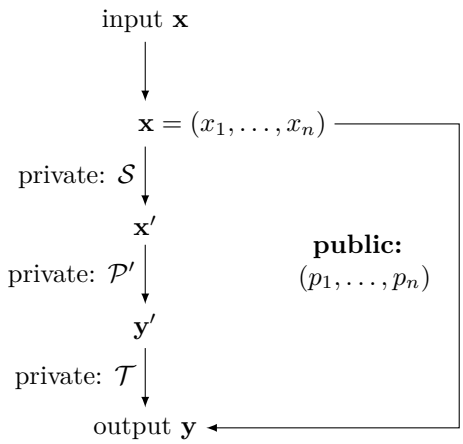


Figure 2.1: Schematic representation of the generic $\mathcal{MQ}$-design with public key $\mathcal{P}$ and private key $(\mathcal{S}, \mathcal{P}', \mathcal{T})$.

the public key $\mathcal{P}$ at $\mathbf{x}$. When used to encrypt, $\mathbf{y} = \mathcal{P}(\mathbf{x})$ becomes the ciphertext. When used to verify a signature, the verifier checks that $\mathcal{P}(\mathbf{x})$ equals the message vector $\mathbf{y} \in \mathbb{F}^m$, for which the signature corresponds to. If $\mathbf{y} = \mathcal{P}(\mathbf{x})$ the signature is accepted, otherwise it is rejected.

Decryption and signature generation

Conceptually, decryption/signing works by reversing the the steps shown Fig. 2.1. Let $\mathbf{y} \in \mathbb{F}^m$ be a ciphertext to be decrypted or a message to be sign. Since the decryptor/signer knows the private key, he can compute this as $\mathbf{x} = \mathcal{S}^{-1} \circ \mathcal{P}'^{-1} \circ \mathcal{T}^{-1}(\mathbf{y})$. As already noted, $\mathcal{S}$ and $\mathcal{T}$ are affine transformations of full rank (with respect to their matrix representations), and can easily be inverted. However, the exact procedure to compute $\mathcal{P}'^{-1}(\mathbf{y}')$ is highly dependent on the central map used in the specific $\mathcal{MQ}$-scheme. We will briefly comment on a few different approaches in the next section.

Additionally, a complicating factor is that $\mathcal{P}'$ might not be injective. In other words, the pre-image of $\mathcal{P}'$ may not necessarily be unique. This is usually not a problem for a signing algorithm, but complicates the process of decryption. For

a ciphertext to be uniquely decrypted, it necessary to add some redundancy in the encryption process so that the correct plaintext can be discerned among the, potentially many, pre-images. However, this is a technicality we will not consider further here, but refer to [WP05b] for details.

2.3 Trapdoor constructions

The big challenge in creating a secure and efficient $\mathcal{MQ}$-system is the construction of the central trapdoor function $\mathcal{P}'$. In Chapter 4 we will describe MQQ-SIG in full detail, so here we just provide a high-level overview on some different approaches to this problem. Generally, there have been two main approaches to constructing the central map $\mathcal{P}'$: "Mixed-Field" (or "Big-Field") constructions and "Single-Field" (or "True") constructions.

In a mixed field construction, $\mathcal{P}$ is created using a bijection in an n-dimensional extension field $\mathbb{E}$ over the ground field $\mathbb{F}$. The first $\mathcal{MQ}$-system C^*, proposed by Matsumoto and Imai [MI88], was of this type. Later, the Hidden Field Equation (HFE) system by Patarin [Pat96] has come to represent another basic trapdoor within this class. The mixed-field construction can briefly be explained as follows. First, we create an invertible map $Q\colon \mathbb{E} \to \mathbb{E}$, and choose a bijection $\phi\colon \mathbb{E} \to \mathbb{F}^n$. Recall that each $\alpha \in \mathbb{E}$ is of the form

$$a_{n-1}t^{n-1} + \cdots + a_1 t + a_0 \text{ with } a_i \in \mathbb{F},$$

and where the operations are performed modulo some irreducible polynomial $\pi(t) \in \mathbb{F}[t]$. Hence, there is always a natural correspondence ϕ, between the field element $\alpha \in \mathbb{E}$ and the vector $\beta = (b_1, \ldots, b_n)$ in the vector space $\mathbb{F}^n$, given by the bijection $a_i \mapsto b_i$. We then define the central map as follows:

$$\mathcal{P}' = \phi \circ Q \circ \phi^{-1}.$$

In C^*, the map Q, is given as

$$Q\colon \mathbf{x} \mapsto \mathbf{x}^{q^\lambda+1},$$

where $\mathbf{x} \in \mathbb{E}$, $q = |\mathbb{F}|$ and $\gcd(q^\lambda + 1, q^n + 1) = 1$. The last criterion is what makes this map invertible, since it guarantees the existence of a μ, such that

$\mu \cdot (q^\lambda + 1) \equiv 1 \pmod{q^n + 1}$. Then, the inversion of $\mathbf{y} = \mathbf{x}^{q^\lambda+1}$ is found simply by raising it to the the power of μ. Altogether, this process is reminiscent to how inversion is performed in RSA, but the hardness assumption is very different. Since the Frobenius automorphism, $\mathbf{x} \mapsto \mathbf{x}^q$, is a linear transformation over the base field $\mathbb{F}$, the central map Q can be represented by quadratic polynomials over $\mathbb{F}$.

HFE was proposed some years after C^* and might initially be seen as a generalization of it. The central map Q is given by

$$Q\colon \mathbf{x} \mapsto \mathbf{y} = \sum_{0 \le i,j < n} a_{ij}\mathbf{x}^{q^i+q^j} + \sum_{0 \le i < n} b_i \mathbf{x}^{q^i} + c,$$

where $a_{ij}, b_j c \in \mathbb{E}$. However, the inversion works very differently from C^*. To invert Q in HFE, it is necessary to use root finding algorithms for polynomials, whereas in C^* it was just raising to an exponent.

In single field constructions, the transformation is defined directly in terms of the base field $\mathbb{F}$, not utilizing any extension fields at all. The two main examples of this class are the "Unbalanced Oil and Vinegar" (UOV) scheme suggested by Patarin et. al [KPG99] and the "Stepwise Triangular System" (STS) due to Wolf and Preneel [WP05a]. In UOV the central polynomials of $\mathcal{P}'$ are of the form

$$p_k(x_1, \ldots, x_n) = \sum_{i=1}^{v}\sum_{j=1}^{n} a_{ij}^{(k)} x_i x_j + \sum_{i=1}^{n} b_i^{(k)} x_j + c^{(k)}.$$

Here $a_{ij}^{(k)}$, $b_i^{(k)}$ and $c^{(k)}$ are all elements in $\mathbb{F}$ and $1 \le k \le v = n - m$. Notice that there are no $x_i x_j$ terms where both $i, j > v$. The variables x_i for $1 \le i \le v$ are called the "vinegar" variables and are combined quadratically. The variables x_j for $v < j \le n$ are called the "oil" variables, and are not combined with each other, but only with other vinegar variables. This is the key point of the UOV design, since it allows for inversion of the system. By assigning random values to the vinegar variables, we obtain a linear system of equations in the oil variables. Such systems can be solved efficiently using Gaussian elimination. UOV is only applicable as a signature scheme since it needs at least twice as many vinegar variables as oil variables in order to be considered secure [WP05b].

2.4 Key sizes and computational aspects

2.4.1 Key sizes

A problem common to all $\mathcal{MQ}$-systems is their public key size. Because the public key is a set of multivariate equations, its size can be very large when compared to other public-key cryptosystems. Recall that a general multivariate quadratic polynomial of n variables over a finite field $\mathbb{F}$, is of the form:

$$p(x_1, \ldots, x_n) = \sum_{1 \leq i \leq j \leq n} a_{ij} x_i x_j + \sum_{i=1}^{n} b_i x_i + c,$$

where $a_{ij}, b_i, c \in \mathbb{F}$. In general, we can potentially have $\frac{n(n+1)}{2}$ different quadratic terms in p. However, for the special case of $\mathbb{F} = \mathrm{GF}(2)$, there will only be $\binom{n}{2}$ quadratic terms, since $x^2 = x$ for all $x \in \mathbb{F}_2$. Hence, the number of potential terms $\tau(n)$ in $p(x_1, \ldots, x_n)$, is given by the formula:

$$\tau(n) = \begin{cases} 1 + n + \dfrac{n(n-1)}{2} = 1 + \dfrac{n(n+1)}{2} & \text{if } \mathbb{F} = \mathrm{GF}(2) \\ 1 + n + \dfrac{n(n+1)}{2} = 1 + \dfrac{n(n+3)}{2} & \text{otherwise.} \end{cases} \tag{2.2}$$

The public key size in $\mathcal{MQ}$-schemes will therefore grow according to the following expression:

$$m \cdot \tau(n) \cdot \log_2 |\mathbb{F}|,$$

where m is the number of polynomials.

Interestingly, some $\mathcal{MQ}$-systems cannot generate all possible terms by their design, making it possible to reduce the size requirements somewhat by choosing an appropriate encoding scheme. Nevertheless, the key sizes will still increase as $O(mn^2)$. Additionally, if a scheme deviates too much from a generic $\mathcal{MQ}$ system, information about the central map might leak through into the public key. This was exactly the case of Patarin's attack [Pat00] on the C^* scheme by Matsumoto and Imai [MI88]. He was able to establish a set of bilinear relations between some of the variables in the public key. This was only possible because of the special properties of the central map $\mathcal{P}' \colon \mathbf{x} \mapsto \mathbf{x}^{q^\lambda + 1}$ (c.f. Section 2.3), which is used in that scheme.

With a common parameter choice like $m \approx 100$ and $n \approx 100$, the public key in $\mathcal{MQ}$-schemes can easily be several hundred kilobytes in size. Compared to a few

thousand bits in systems like RSA, we see that cryptography based on multivariate cryptography carries an extra burden on its storage and memory requirements.

2.4.2 Efficiency of cryptography based on $\mathcal{MQ}$-trapdoors

There are two "directions" to consider when discussing the efficiency of a public-key scheme: the encrypting/verification direction, and the decrypting/signing direction. In a $\mathcal{MQ}$-system this corresponds to applying the functions $\mathcal{T} \circ \mathcal{P}' \circ \mathcal{S}$ and $\mathcal{S}^{-1} \circ \mathcal{P}'^{-1} \circ \mathcal{T}^{-1}$, respectively. However, in reality the first function represents the public key of the scheme, so the internal components are not known. Externally, only their composition is visible, which is of course the system of multivariate polynomial equations $\mathcal{P}$. Thus, the two directions are fundamentally different from an implementation point of view. The encryption/verification direction is the evaluation of m multivariate quadratic polynomials, while the decrypting/signing direction is the application of two linear transformations and one quadratic map. This is in contrast to the public-key schemes we use today, where both directions are quite similar. There are of course differences in these as well, for example many implementations are optimized to favor particularly efficient operation in one direction at the expense of the other, but they are based on the same underlying algorithms and modular arithmetic. For $\mathcal{MQ}$-schemes, the two directions can require very different implementations.

We consider the encryption/verification direction first. From Equation 2.2, we have that each polynomial p_i in the public key has $\tau(n) = O(n^2)$ terms. Hence, to evaluate it at the value $\mathbf{x} \in \mathbb{F}^n$, we need to perform $O(n^2)$ multiplications and additions in the finite field $\mathbb{F}$. This yields a total of $O(mn^2)$ additions and multiplications for the full public key. Strategies for how the evaluation of the public key can be efficiently implemented are discussed in [BBG07].

We now turn to the decryption/signing direction. Recall that $\mathcal{S}$ and $\mathcal{T}$ are affine bijections on $\mathbb{F}^n$ and $\mathbb{F}^m$, respectively. They are usually implemented as multiplications with $n \times n$ and $m \times m$ matrices over the finite field $\mathbb{F}$. So the time complexity of the secret map is $O(n^2 + m^2)$ multiplications in $\mathbb{F}$, plus whatever time needed to invert the central map $\mathcal{P}$. This last part is clearly dependent on the specific trapdoor function used, so it is difficult to discuss its complexity in general.

CHAPTER 3

Quasigroups

As described in Chapter 2, the main distinctive feature of the various $\mathcal{MQ}$-schemes is how they construct the central map $\mathcal{P}'$. We also noted that essentially, there are only four basic trapdoor designs known. Other schemes are simply variations on these four base classes. Unfortunately, most of them have been broken in their basic forms.

Recently, a completely new approach to creating $\mathcal{MQ}$-trapdoors was proposed by Gligoroski et al. in 2008 [GMK08b]. Their idea was to use the algebraic structures called *quasigroups* to form the basis for the central invertible map. Shortly thereafter, they introduced a public-key cryptosystem based on this idea, called MQQ [GMK08a], and later a signature scheme, called MQQ-SIG [GØJ+11]. The last "Q" in those acronyms stands for "Quasigroup".

In this chapter we present the necessary theoretical background on quasigroups, in order to be able to describe the MQQ-SIG scheme in Chapter 4. We refer to [Smi07] for a more thorough treatment of quasigroup theory.

3.1 Basic definitions

Definition 3.1. A *quasigroup* $(Q, *)$ is a set Q together with a binary operation $*$, such that for each u, v in Q there exists unique elements x, y in Q satisfying

$$\begin{aligned} u * x &= v, \\ y * u &= v. \end{aligned} \tag{3.1}$$

The number of elements in the set Q is called the *order* of the quasigroup.

The unique solutions x and y defines two binary operators upon the quasigroup, called the *left* and *right division*[*] respectively. They are denoted $\backslash$ and $/$ and defined by the relations

$$u * x = v \Leftrightarrow x = u\backslash v \Leftrightarrow u = v/x. \tag{3.2}$$

It is easy to see that this makes $(Q, \backslash)$ and $(Q, /)$ quasigroups in their own rights. The operators $\backslash$ and $/$ satisfy the following identities[†]:

Proposition 3.2. *The quasigroup $(Q, *)$ together with the binary operators $\backslash$ and $/$ satisfies the identities*

$$\begin{aligned} u * (u\backslash v) &= v, \\ (v/u) * u &= v, \\ u\backslash(u * v) &= v, \\ (v * u)/u &= v \end{aligned} \tag{3.3}$$

for all $u, v \in Q$.

Proof. The first two identities are simply the definitions of being a left and right division in a quasigroup. For the third, note that $u\backslash(u*v)$ is a solution of $u*x = u*v$, for all $u, v \in Q$. Simultaneously, we see that also v is a solution of this equation. Hence, by the uniqueness of the left division, $u\backslash(u * v) = v$. The last case is proved analogously. □

---

[*]Sometimes also called the *left* and *right parastrophe*.

[†]In fact, this proposition could be taken as the definition of the quasigroups $(Q, *), (Q, \backslash)$ and $(Q, /)$, derived from the algebra $(Q, *, \backslash, /)$.

Definition 3.3. An *identity* element in a quasigroup Q is an element e such that $e * x = x = x * e$ holds for all x in Q. A quasigroup with an identity element is called a *loop*.

Example 3.4. Each group is a quasigroup, with $x/y = xy^{-1}$ and $x\backslash y = x^{-1}y$. Conversely, every (non-empty) associative quasigroup is a group. △

Example 3.5. $(\mathbb{Z}, -)$ is a non-associative quasigroup. △

Example 3.6. A *Latin square* is an $n \times n$ square filled with n different symbols, arranged such that each symbol occurs exactly once in each column and in each row. The multiplication table of a (finite) quasigroup is a Latin square. Conversely; each Latin square yields the multiplication table of a quasigroup. This is easily seen to be true, since for any Latin square we can add a left border column and a top border row and label them with the symbols $1, \ldots, n$. Each symbol corresponds to an element in the quasigroup. In Figure 3.1 we see an example of a 5×5 Latin square on the left and a corresponding quasigroup multiplication table on the right, with $4 * 3 = 2$, and so on. △

1	3	4	2	5
3	2	1	5	4
4	1	5	3	2
5	4	2	1	3
2	5	3	4	1

$*$	1	2	3	4	5
1	1	3	4	2	5
2	3	2	1	5	4
3	4	1	5	3	2
4	5	4	2	1	3
5	2	5	3	4	1

Figure 3.1: A Latin square and a corresponding finite quasigroup.

Definition 3.7. Let $(Q_1, *_1)$ and $(Q_2, *_2)$ be quasigroups. A mapping

$$\alpha \colon (Q_1, *_1) \to (Q_2, *_2)$$

is called a *homomorphism* if for all $u, v \in Q_1$

$$\alpha(u *_1 v) = \alpha(u) *_2 \alpha(v).$$

If α is bijective we call it an *isomorphism* and say that Q_1 and Q_2 are *isomorphic*.

Within quasigroup theory, the common notion of homomorphism's is often too strong. A weaker requirement is that of a homotopy:

Definition 3.8. Let $(Q_1, *_1)$ and $(Q_2, *_2)$ be quasigroups. A triple of maps (α, β, γ)

$$\alpha, \beta, \gamma \colon (Q_1, *_1) \to (Q_2, *_2)$$

is called a *homotopy* if for all $u, v \in Q_1$

$$\alpha(u) *_2 \beta(v) = \gamma(u *_1 v).$$

The triple is an *isotopy* if the maps α, β, γ are bijective. We then say that Q_1 and Q_2 are *isotopic*, and denote it $Q_1 \sim Q_2$.

Proposition 3.9. *Let $(Q_1, *_1)$ be a quasigroup, and let Q_2 be a set such that $|Q_1| = |Q_2|$. Let α, β, γ be bijective maps: $Q_2 \to Q_1$. Then $(Q_2, *_2)$, defined by*

$$u *_2 v = \gamma^{-1}\left(\alpha(u) *_1 \beta(v)\right),$$

for all u, v in Q_2, is a quasigroup and (α, β, γ) is an isotopy between $(Q_2, *_2)$ and $(Q_1, *_1)$.

Proof. We need to prove the existence of unique left and and right divisors in Q_2. Since both cases are handled completely analogous, we only prove the existence of a unique left divisor.

Let u and w be two elements in the set Q_2. We want to find a unique $v \in Q_2$, such that $u *_2 v = w$. Let $u' = \alpha(u)$ and $w' = \gamma(w)$, be the two images of u and w in Q_1, under the bijections α and γ, respectively. Since Q_1 is a quasigroup, there exists a unique $v' \in Q_1$ such that $u' *_1 v' = w'$. Let v be its corresponding element in Q_2 under the inverse map β^{-1}. Then

$$\begin{aligned} w &= \gamma^{-1}(w') \\ &= \gamma^{-1}(u' *_1 v') \\ &= \gamma^{-1}(\alpha(u) *_1 \beta(v)) \\ &= u *_2 v. \end{aligned}$$

Since α, β and γ all are bijections, uniqueness of $*_2$ follows from the uniqueness of $*_1$ in Q_1. The isotopy follows directly from the definition. □

3.2 Multivariate quadratic quasigroups

In this section we will define the special class of quasigroups known as *multivariate quadratic quasigroups*. First, we recall some facts about k-ary Boolean functions.

Definition 3.10. A map $\alpha\colon \mathbb{F}_2^k \to \mathbb{F}_2$ is called a *k-ary Boolean function*.

It is known that every Boolean function can be represented as a Boolean polynomial. That is, there exists an isomorphism between the ring of Boolean functions and the ring of Boolean polynomials. We state this fact without proof in the following lemma. A a proof can be found in [Chr09, Thm. 2.3].

Lemma 3.11. *Every k-ary Boolean function can be uniquely represented as a Boolean polynomial from $\mathbb{F}_2[x_1, \ldots, x_k]$.* □

The unique polynomial representing a k-ary Boolean function $f(x_1, \ldots, x_k)$, also known as its algebraic normal form (ANF), can in general be written as

$$ANF(f) = a_0 + \sum_{1\le i\le k} a_i x_i + \sum_{1\le i<j\le k} a_{i,j} x_i x_j + \sum_{1\le i<j<s\le k} a_{i,j,s} x_i x_j x_s + \ldots$$

where all coefficients $a_0, a_i, a_{i,j}, \ldots$ are in $\mathbb{F}_2$. Because of the unique correspondence between a k-ary Boolean function f, and its ANF-form, we find it convenient to identify it directly as if it were a polynomial $f(x_1, \ldots, x_k) \in \mathbb{F}_2[x_1, \ldots, x_k]$.

Now, given a function $\alpha\colon \mathbb{F}_2^k \times \mathbb{F}_2^k \to \mathbb{F}_2^k$, we can represent the k-dimensional vector $\alpha = (f_1, \ldots, f_k)$, consisting of $2k$-ary Boolean functions f_i, as a vector of Boolean polynomials $(f_1, f_2, \ldots, f_k) \in (\mathbb{F}_2[x_1, \ldots, x_{2k}])^k$.

Definition 3.12. Let α be a function

$$\alpha\colon \overbrace{\mathbb{F}_2^k \times \ldots \times \mathbb{F}_2^k}^{m} \to \mathbb{F}_2^k$$

and let $(f_1, f_2, \ldots, f_k) \in (\mathbb{F}_2[x_1, \ldots, x_{2m}])^k$ be its representation in Boolean polynomials. Then we say that α *is of degree d* if $d = \max\{\deg(f_i), i = 1, \ldots, k\}$.

Let $(Q, *)$ be a finite quasigroup of order 2^d. We can identify every element in Q with a unique element in $\mathbb{F}_2^d$. Then the binary operator $*$ can be seen as a Boolean map $*\colon \mathbb{F}_2^d \times \mathbb{F}_2^d \to \mathbb{F}_2^d$ defined as

$$* : (u, v) \mapsto w,$$

where $u, v, w \in \mathbb{F}_2^d$ satisfy $u * v = w$. Thus, from the above we can state:

Theorem 3.13. *For every quasigroup $(Q, *)$ of order 2^d and for each bijection $Q \to \mathbb{F}_2^d$, there is a unique map $*_q \colon \mathbb{F}_2^d \times \mathbb{F}_2^d \to \mathbb{F}_2^d$, consisting of d uniquely determined $2d$-ary Boolean functions $f_1, \ldots, f_d$, such that for each $u, v, w \in Q$*

$$u * v = w$$

if and only if

$$*_q(x_1, \ldots, x_d, y_1, \ldots, y_d) = \begin{pmatrix} f_1(x_1, \ldots, x_d, y_1, \ldots, y_d) \\ \vdots \\ f_d(x_1, \ldots, x_d, y_1, \ldots, y_d) \end{pmatrix},$$

where $x_1, \ldots, x_d$ and $y_1, \ldots, y_d$ are the binary representations of u and v respectively. □

The representation of quasigroups as a vector of Boolean functions, gives us a way of classifying them according to their polynomial degrees. The degrees of quasigroups of order $2^d, d \geq 4$, will in generally be higher than two when generated at random. As explained in Chapter 2, public-key cryptography based on multivariate equations are usually confined to quadratic polynomials, since any degrees higher than will lead to extremely large keys. We now define the important class of quasigroups which are represented by multivariate quadratic polynomials.

Definition 3.14. A quasigroup $(Q, *)$ of order 2^d is called a *Multivariate Quadratic Quasigroup (MQQ) of type* $Quad_{d-k}Lin_k$ if exactly $d - k$ of the polynomials f_i are of degree 2 (i.e. quadratic) and k of them are of degree 1 (i.e. linear), where $0 \leq k < d$.

Example 3.15. Let $(Q, *)$ be the quasigroup of order $2^3 = 8$ defined by the following multiplication table

$*$	0	1	2	3	4	5	6	7
0	1	0	6	7	2	3	5	4
1	3	6	4	1	0	5	7	2
2	0	5	7	2	3	6	4	1
3	2	3	5	4	1	0	6	7
4	7	1	3	6	4	1	0	5
5	5	4	1	0	6	7	2	3
6	6	7	2	3	5	4	1	0
7	4	1	0	5	7	2	3	6

It can be represented by a corresponding vector of Boolean functions

$$*_q(x_1, x_2, x_3, x_4, x_5, x_6) = \begin{pmatrix} x_1x_6 + x_2x_6 + x_3x_6 + x_1 + x_5 \\ x_1x_5 + x_1 + x_3 + x_4 + x_5 \\ x_1x_5 + x_2 + x_4 + x_5 + x_6 + 1 \end{pmatrix},$$

where

$$\begin{aligned} f_1 &= x_1x_6 + x_2x_6 + x_3x_6 + x_1 + x_5 \\ f_2 &= x_1x_5 + x_1 + x_3 + x_4 + x_5 \\ f_3 &= x_1x_5 + x_2 + x_4 + x_5 + x_6 + 1, \end{aligned}$$

is the ANF of the Boolean functions constituting the function $*_q$. Since they are all of degree two, this is an MQQ △

It is well known from linear algebra that a function defined as $T(\mathbf{x}) = A \cdot \mathbf{x} + \mathbf{c}$, where A is an invertible $n \times n$ matrix over $\mathbb{F}_2$ and $\mathbf{c} \in \mathbb{F}_2^n$, is a permutation of the vector space $\mathbb{F}_2^n$. By Lemma 3.11, T can also be represented as a vector $(f_1, f_2, \ldots, f_n)$, of linear Boolean functions. We call such T a *linear permutation*. A useful property of linear permutations, and one that we will need shortly, is given in the following lemma. However, the proof is somewhat technical and not very relevant for our purposes, so we will skip it. A proof can be found in [Chr09, Lemma 2.25].

Lemma 3.16. *Let α be a function*

$$\alpha\colon \overbrace{\mathbb{F}_2^k \times \ldots \times \mathbb{F}_2^k}^{m} \to \mathbb{F}_2^k$$

of degree $d > 0$, and $\mathcal{L}$ be a linear permutation of $\mathbb{F}_2^k$. Then the maps

$$\alpha' \colon \mathbf{x}_1, \ldots, \mathbf{x}_m \mapsto \alpha\left(\mathbf{x}_1, \ldots, \mathcal{L}(\mathbf{x}_r), \ldots, \mathbf{x}_m\right)$$

where $1 \leq r \leq m$, and

$$\alpha'' \colon \mathbf{x}_1, \ldots, \mathbf{x}_m \mapsto \mathcal{L}\left(\alpha\left(\mathbf{x}_1, \ldots, \mathbf{x}_m\right)\right)$$

are also of degree d. □

Definition 3.17. Let Q_1, Q_2 be quasigroups upon $\mathbb{F}_2^d$. We say that Q_1 and Q_2 are *linearly isotopic* if there exists three linear permutations $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$, which forms an isotopy $(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3) \colon Q_1 \to Q_2$. If $\mathcal{L}_1 = \mathcal{L}_2 = \mathcal{L}_3$ we say that Q_1 and Q_2 are *linearly isomorphic*.

Theorem 3.18. *Let $(Q_1, *_1)$ and $(Q_2, *_2)$ be quasigroups upon $\mathbb{F}_2^d$. Define the relation $\sim$ so that $Q_1 \sim Q_2$ if and only if Q_1 is linearly isotopic to Q_2. Then $\sim$ is an equivalence relation on the set of quasigroups.*

Proof. For every quasigroup Q, $(\mathrm{id}, \mathrm{id}, \mathrm{id})$ is a linear isotopy from Q to itself, so $\sim$ is reflexive.

Suppose $Q_1 \sim Q_2$, so there exists a linear isotopy $(\alpha, \beta, \gamma) \colon Q_1 \to Q_2$. In particular we have that $u *_1 v = \gamma^{-1}\left(\alpha(u) *_2 \beta(v)\right)$. Then the inverses of α, β and γ represents a linear isotopy from Q_2 to Q_1, that is,

$$\begin{aligned} \alpha^{-1}(u) *_1 \beta^{-1}(v) &= \gamma^{-1}\left(\alpha\left(\alpha^{-1}(u)\right) *_2 \beta\left(\beta^{-1}(v)\right)\right) \\ &= \gamma^{-1}\left(u *_2 v\right), \end{aligned}$$

for $u, v \in Q_2$. So $\sim$ is symmetric.

Finally, assume $Q_1 \sim Q_2$ and $Q_2 \sim Q_3$, then there exists linear isotopies $(\alpha, \beta, \gamma) \colon Q_1 \to Q_2$ and $(\alpha', \beta', \gamma') \colon Q_2 \to Q_3$. Again we note that

$$\begin{aligned} u *_1 v &= \gamma^{-1}\left(\alpha(u) *_2 \beta(v)\right) \\ u *_2 v &= \gamma'^{-1}\left(\alpha'(u) *_3 \beta'(v)\right), \end{aligned}$$

for any $u, v \in \mathbb{F}_2^d$. We claim that the triple $(\alpha'', \beta'', \gamma'')$, given by $\alpha'' = \alpha'\alpha$, $\beta'' = \beta'\beta$ and $\gamma'' = \gamma'\gamma$, is a linear isotopy from Q_1 to Q_3. For let $u, v \in Q_1$, then

$$\begin{aligned}
\gamma''(u *_1 v) &= \gamma'' \left(\gamma^{-1}\left(\alpha(u) *_2 \beta(v)\right)\right) \\
&= \gamma''\gamma^{-1}\left(\alpha(u) *_2 \beta(v)\right) \\
&= \gamma''\gamma^{-1}\left(\gamma'^{-1}\left(\alpha'\left(\alpha(u)\right) *_3 \beta'\left(\beta(v)\right)\right)\right) \\
&= \gamma''\gamma^{-1}\gamma'^{-1}\left(\alpha'\alpha(u) *_3 \beta'\beta(v)\right) \\
&= \alpha''(u) *_3 \beta''(v),
\end{aligned}$$

so $\sim$ is also transitive. □

We call the equivalence class of a linear isotopy a *linear isotopy class*. It is interesting to know which characteristics of a MQQ that are invariant under a linear isotopy. That is, which properties are shared among all the quasigroups in a linear isotopy class? This question is studied in more detail in Chapter 5, but we can state one easy fact here.

Corollary 3.19. *Linearly isotopic Boolean quasigroups have the same degree.*

Proof. Immediate from Lemma 3.16. □

CHAPTER 4

MQQ-SIG

In 2008, Gligoroski, Markovski and Knapskog [GMK08b] proposed a new way of creating $\mathcal{MQ}$-trapdoors, using *quasigroup string transformations*. Basically, a quasigroup string transformation treats a bit string $\mathbf{z} \in \{0,1\}^n$, as consisting of $k = \frac{n}{d}$ equally sized substrings $\mathbf{z}_i$. Each of these substrings can be regarded as elements of a quasigroup $(Q, *)$, of 2^d elements. Thus, it is possible to define a transformation on $\mathbf{z}$, based on some application of the quasigroup operation to its substrings $\mathbf{z}_i$. This was exactly the idea of Gligoroski et al., when they devising their new trapdoor design.

Later they showed how this trapdoor construction could be used to build a public-key cryptography scheme, which they simply called MQQ* [GMK08a]. However, shortly after its publication the MQQ cipher was broken both by Perret† using a Gröbner basis approach (which we will return to in Chapter 5); and Mohamed et al., using the MutantXL algorithm [MDBW09].

Recently, Gligoroski et al. proposed a signature scheme [GØJ+11] based on the main ideas from the MQQ cryptosystem, called MQQ-SIG. Besides being a signature scheme, the main difference that separates MQQ-SIG from the MQQ cipher, is that

*To avoid the potential confusion with the abbreviation we so far have used to denote a multivariate quadratic quasigroup, we will always qualify the MQQ cipher as "the cryptosystem MQQ", "the MQQ cipher" or something similar.

†In a personal e-mail correspondence with one of the designers, according to [FØPG10].

it only utilize one quasigroup for the central map $\mathcal{P}$, while the original MQQ cipher used several. Additionally, to counter the previously successful attacks, MQQ-SIG applies a modification known as the *minus-modifier* (see [WP05b]), which removes some equations from the public key. In particular, the first $\frac{n}{2}$ equations are removed. They also proved that MQQ-SIG is existentially unforgable under a chosen-message attack (or CMA-secure for short) in the random oracle model.

This chapter presents the full details of the MQQ-SIG scheme. We explain both the operation of the system, and the rationale that went into designing it, according to its designers [GØJ+11]. Additionally, we discuss some computational and implementation aspects.

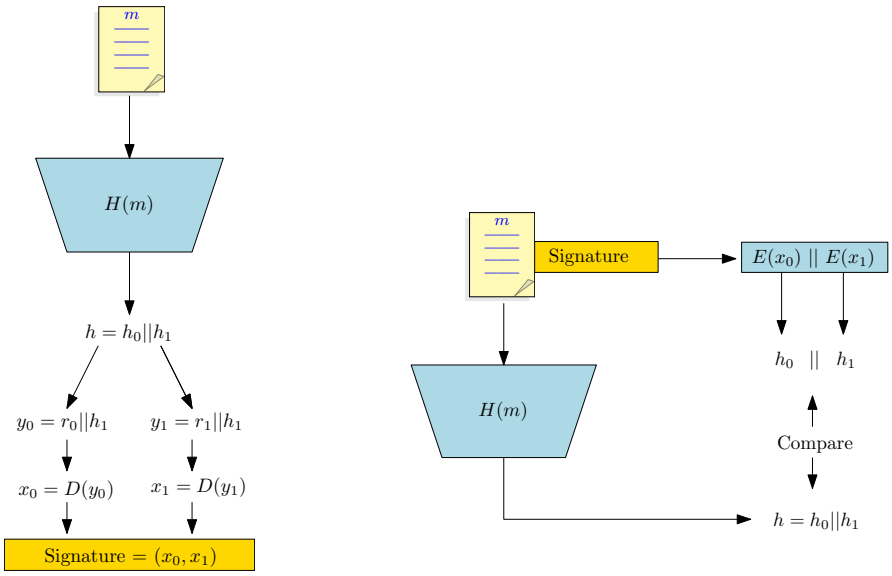
4.1 High level description

MQQ-SIG is, like other $\mathcal{MQ}$-schemes, based on two bijective linear/affine transformations and one central multivariate quadratic map:

$$\mathcal{P} = \mathcal{S} \circ \mathcal{P}' \circ \mathcal{S}' \colon \{0,1\}^n \to \{0,1\}^n. \tag{4.1}$$

Here, $\mathcal{S}'$ is a bijective affine transformation defined as $\mathcal{S}'(\mathbf{x}) = \mathcal{S}(\mathbf{x}) + \mathbf{v}$, where $\mathcal{S}$ is given by a bijective linear transformation $\mathcal{S}(\mathbf{x}) = S \cdot \mathbf{x}$. We will describe the construction and operation of the central map $\mathcal{P}'$, in further detail in Section 4.2 and the linear transformation $\mathcal{S}$ in Section 4.3. Here, it suffices to know that they are efficiently invertible transformations. As mentioned in the introduction, MQQ-SIG employs a common modification of the standard $\mathcal{MQ}$-structure, known as the "minus"-modifier, which discards half of the public polynomials in the public key (we refer to [WP05b], for a description of this, and other modifications that can be done to $\mathcal{MQ}$-schemes). In other words, the public key consists of only half of the equations generated by the function in Equation 4.1. The operation of the MQQ-SIG as a digital signature system is specified in Algorithm 4.1.

To sign a message m in MQQ-SIG, the message is first hashed by some standard hash function H. The message digest is then split into two halves which are concatenated with two random values. The two values obtained after concatenation are then fed to the inverse transformations $\mathcal{S}^{-1}$, ${\mathcal{P}'}^{-1}$ and ${\mathcal{S}'}^{-1}$. The output is the signature of the message. The signing process is illustrated in Figure 4.1a.



(a) Signing **(b)** Verification

Figure 4.1: A graphical presentation of the signing and verification process in MQQ-SIG. D denotes the function composition: $D = \mathcal{S}'^{-1} \circ \mathcal{P}'^{-1} \circ \mathcal{S}^{-1}$.

To verify a message-signature pair, the message is first hashed with the standard hash function. Then the public key is evaluated on each element of the pair which makes up the signature. If the message digest and the concatenated result of the two public key evaluations are equal, the signature is deemed valid. Otherwise, it is rejected. The verification process is illustrated in Figure 4.1b.

Algorithm 4.1 The MQQ-SIG Signature Scheme

Key Setup
Let Alice be a user of the MQQ-SIG scheme. First she generates a MQQ according to Algorithm 4.2. This provides her with a description of the central map $\mathcal{P}'$ and its inverse $\mathcal{P}'^{-1}$. Then she generates an invertible $n \times n$ Boolean matrix S according to the description in Section 4.3. Additionally she picks a hash function $H : \{0,1\}^* \to \{0,1\}^n$. The hash function is a standard public parameter of the MQQ-SIG scheme. Alice now performs the following steps:

1. computes a vector $\mathbf{y}$ of n quadratic multivariate polynomials $P_i(x_1, \ldots, x_n)$ in n variables as:
$$\mathbf{y} = \mathcal{S}(\mathcal{P}(\mathcal{S}'(\mathbf{x}))),$$
where $\mathbf{x} = (x_1, \ldots, x_n)$, and $\mathcal{S}$ and $\mathcal{S}'$ are defined by the matrix S;
2. defines the polynomial vector E as the last half of $\mathbf{y}$. That is, the elements of E consists of the polynomials $P_i(x_1, \ldots, x_n)$ for $i = 1 + \frac{n}{2}, \ldots, n$.

The public key is the vector E. Alice's private key is the triple $(\mathcal{S}^{-1}, \mathcal{P}'^{-1}, \mathcal{S}'^{-1})$.

Signature Generation
To sign a message $m \in \{0,1\}^*$, Alice performs the following steps:

1. computes $h = h_0||h_1 \leftarrow H(m)$, where h_0 and h_1 both are $\frac{n}{2}$ bits long;
2. generates two random $\frac{n}{2}$-bit values, r_0 and r_1;
3. set $\mathbf{y_0} = r_0||h_0$ and $\mathbf{y_1} = r_1||h_1$;
4. computes $\mathbf{x_0} = \mathcal{S}'^{-1}(\mathcal{P}'^{-1}(\mathcal{S}^{-1}(\mathbf{y_0})))$ and $\mathbf{x_1} = \mathcal{S}'^{-1}(\mathcal{P}'^{-1}(\mathcal{S}^{-1}(\mathbf{y_1})))$.

The digital signature is the pair $(\mathbf{x_0}, \mathbf{x_1})$.

Signature Verification
To verify a message-signature pair $(m, (\mathbf{x_0}, \mathbf{x_1}))$, Bob performs the following steps:

1. compute $h = h_0||h_1 \leftarrow H(m)$;
2. compute $\mathbf{z_0} \leftarrow E(\mathbf{x_0})$ and $\mathbf{z_1} \leftarrow E(\mathbf{x_1})$.

He accepts the signature if $\mathbf{z_0} = h_0$ and $\mathbf{z_1} = h_1$, otherwise he rejects.

4.2 The central map $\mathcal{P}'$

4.2.1 Construction of the quasigroup

The most distinguishing factor of the MQQ-SIG scheme is how it constructs its central map $\mathcal{P}'$. It is at this stage the theory of multivariate quadratic quasigroups is utilized to create a trapdoor which is efficiently invertible.

First, recall from Section 3.2 that any quasigroup of order 2^d can be uniquely represented as a d-element vector consisting of $2d$-ary Boolean functions. If at least one of these functions are of degree 2, and the rest of degree 2 or less, the quasigroup is said to be multivariate quadratic (MQQ). Thus, when we consider the operation of a given multivariate quadratic quasigroup, it suffices to regard it as a mapping in the vector space $\mathbb{F}_2^d$.

The following Lemma provides sufficient conditions for a quasigroup to be an MQQ.

Lemma 4.1. **([CKG10])** *Let $A_1' = (f_{ij})$, $A_2' = (g_{ij})$ be two $d \times d$ matrices of linear Boolean expressions in the variables $x_1, \ldots, x_d$ and $y_1, \ldots, y_d$ respectively. Let $\mathbf{c}$ be a binary column vector of d elements. If the following two conditions are fulfilled:*

$$\det(A_1') = \det(A_2') = 1,$$
$$A_1' \cdot (y_1, \ldots, y_d)^T + (x_1, \ldots, x_d)^T = A_2' \cdot (x_1, \ldots, x_d)^T + (y_1, \ldots, y_d)^T,$$

then the function

$$*_{vv}(x_1, \ldots, x_d, y_1, \ldots, y_d) = A_1' \cdot (y_1, \ldots, y_d)^T + (x_1, \ldots, x_d)^T + \mathbf{c}',$$

defines a quasigroup $(Q, *_{vv})$ of order 2^d that is an MQQ. □

In [CKG10, Thm. 11] the conditions in Lemma 4.1 were refined to allow for a more constructive statement on how to build MQQ's. In particular, they show that the matrices A_1, A_2, are of the form

$$\begin{aligned} A_1' &= I_d + \left((a_1^{ij}, \ldots, a_d^{ij}) \cdot \mathbf{x}\right)_{d \times d}, \\ A_2' &= I_d + \left((b_1^{ij}, \ldots, b_d^{ij}) \cdot \mathbf{y}\right)_{d \times d}, \end{aligned} \tag{4.2}$$

where $a_k^{ij}, b_k^{ij} \in \mathbb{F}_2$, and $\mathbf{x} = (x_1, \ldots, x_d), \mathbf{y} = (y_1, \ldots, y_d) \in \mathbb{F}_2^d$.

To construct the MQQ's in the MQQ-SIG scheme, Gligoroski et al. [GØJ+11] defines the quasigroup operation as follows:

$$\mathbf{x} * \mathbf{y} = B \cdot U(\mathbf{x}) \cdot A_2 \cdot \mathbf{y} + B \cdot A_1 \cdot \mathbf{x} + \mathbf{c}, \tag{4.3}$$

where $\mathbf{x} = (x_1, \ldots, x_d)$, $\mathbf{y} = (y_1, \ldots, y_d)$, and A_1, A_2 and B are invertible $d \times d$ matrices over $\mathbb{F}_2$, and the vector $\mathbf{c}$ is an element in $\mathbb{F}_2^d$. The matrix $U(\mathbf{x})$ is an upper triangular matrix with all diagonal elements equal to 1, and where the elements above the main diagonal are linear expressions in the variables $x_1, \ldots, x_d$.

The matrices A_1, A_2 and B, together with the vector $\mathbf{c}$, can all be generated by a uniformly random process. The matrix $U(\mathbf{x})$ is computed as a matrix of column vectors as follows:

$$U(\mathbf{x}) = I_d + \begin{pmatrix} U_1 \cdot A_1 \cdot \mathbf{x} & \cdots & U_d \cdot A_1 \cdot \mathbf{x} \end{pmatrix}, \tag{4.4}$$

where the U_i's are strictly upper triangular matrices having all elements in the rows $\{i, \ldots, d\}$ zero. The elements in the rows $\{1, \ldots, i-1\}$ (and which lays above the main diagonal) may be either 0 or 1, generated uniformly at random. In particular, U_1 is the all zero matrix.

Theorem 4.2. *The function $*$ defined in Equation 4.3 represents a multivariate quadratic quasigroup of order 2^d.*

Proof. Let us first consider Equation 4.3 without A_1, A_2 and B, that is, we define the binary operator $*'$:

$$\mathbf{x} *' \mathbf{y} = U(\mathbf{x}) \cdot \mathbf{y} + \mathbf{x} + \mathbf{c}',$$

where we let $\mathbf{c}' = B^{-1} \cdot \mathbf{c}$. Note that $U(\mathbf{x})$ is a special form of the matrices in Equation 4.2, and in particular, $\det(U(\mathbf{x})) = 1$. Hence, this modified expression is a multivariate quadratic quasigroup according to Lemma 4.1.

Now, since A_1, A_2 and B are all bijections, Proposition 3.9 gives that $(\mathbb{F}_2^d, *)$ is a quasigroups, and (A_1, A_2, B) is an isotopy from $(\mathbb{F}_2^d, *')$ to $(\mathbb{F}_2^d, *)$. Since the degrees of quasigroups are kept under linear isotopies (cf. Proposition 3.19), $(\mathbb{F}_2^d, *)$ is an MQQ. □

Example 4.3. Referring back to Example 3.15 in Chapter 3, we had a quasigroup $(Q, *)$, which could be represented by the operation

$$*_q = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} = \begin{pmatrix} x_1x_6 + x_2x_6 + x_3x_6 + x_1 + x_5 \\ x_1x_5 + x_1 + x_3 + x_4 + x_5 \\ x_1x_5 + x_2 + x_4 + x_5 + x_6 + 1 \end{pmatrix},$$

where f_1, f_2 and f_3, were the ANF of the corresponding Boolean functions. Since they are all of degree two, this is an MQQ of order $2^3 = 8$. Moreover, it is actually constructed according to Equation 4.3, where

$$A_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \qquad A_2 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \qquad U(\mathbf{x}) = \begin{pmatrix} 1 & x_1 & 0 \\ 0 & 1 & x_1 + x_2 + x_3 \\ 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{c} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix},$$

This is an example of the kinds of quasigroups used in MQQ-SIG (albeit with $d = 3$, which is too low to be used in a real system). △

In addition to the definition in Equation 4.3, the designers of MQQ-SIG also required the MQQ to satisfy the following two con:

$$\forall i \in \{1, \ldots, d\}\,,\ \mathrm{Rank}(\mathbf{B}_{f_i}) \geq 2d - 4, \tag{4.5a}$$

$$\exists j \in \{1, \ldots, d\}\,,\ \mathrm{Rank}(\mathbf{B}_{f_j}) = 2d - 2, \tag{4.5b}$$

where $\mathbf{B}_{f_i}$ is a $2d \times 2d$ symmetric Boolean matrix, determined by the i'th polynomial f_i in the MQQ, as

$$\mathbf{B}_{f_i} = \begin{pmatrix} 0 & M \\ M^T & 0 \end{pmatrix}, \tag{4.6}$$

where each entry in the $d \times d$ matrix M, corresponds to a term $x_s y_t$, $1 \leq s, t \leq d$, in the original polynomial f_i. If the term $x_s y_t$ is present in f_i then the entry in row s, column t, in M is 1. Otherwise it is 0. In other words, if $\mathbf{B}_{f_i} = (b_{jk})_{2d \times 2d}$, then $b_{j,d+k} = b_{d+k,j} = 1$ if and only if $x_j y_k$ is a term in f_i.

According to Gligoroski et al. [GØJ+11], was the rationale for including the constraints in Equation 4.5a and 4.5b, to thwart the so-called MinRank attacks [BFS99, FLDVP08]. The MinRank problem is the following: given a sequence of matrices $(M_1, \ldots, M_n)$ over some finite field and an integer $r < n$, find a linear combination of the matrices $M_1, M_2, \ldots, M_n$, such that $\text{Rank}(\sum_{i=1}^{n} \lambda_i M_i) \leq r$. For special instances of this problem there exists efficient methods to solve the MinRank problem. These solutions can then further be used to break some $\mathcal{MQ}$-systems with certain properties [GC00]. We will return to the purpose of these constraints in Chapter 5, where we analyze the parameters of MQQ-SIG.

4.2.2 Operation of the central map $\mathcal{P}'$

We now explain how quasigroup string transformations can be used to construct the central map $\mathcal{P}'$. In MQQ-SIG the parameter d is set to 8, so the size of the quasigroup is 2^8. Recall that $\mathcal{P}'$ is a function $\mathcal{P}' \colon \{0,1\}^n \to \{0,1\}^n$, mapping n bits of input to n bits of output. It is required that n be a multiple of 8.

Consider the bit string $\mathbf{x} = X = X_1 X_2 \cdots X_k$ of $k = \frac{n}{8}$ equally sized bit strings X_i, based on the original input string x. As explained in Chapter 3, we can identify each X_i with an element in the vector space $\mathbb{F}_2^8$. Hence, we can view X as a string of elements from the quasigroup $(\mathbb{F}_2^8, *)$ defined in Equation 4.3. The quasigroup string transformation, turns X into to the output bit string $Y = Y_1 Y_2 \cdots Y_k$, by a process illustrated in Figure 4.2.

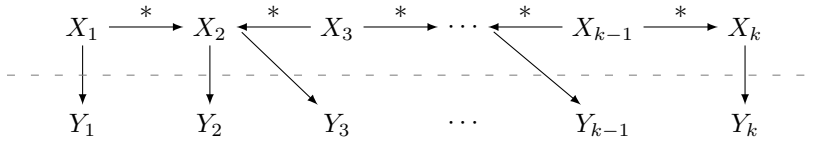


Figure 4.2: A graphical presentation of the construction of the central map $\mathcal{P}'$.

In words: the quasigroup operation $*$, is applied successively to each consecutive pair of vectors X_i, and X_{i-1}. The procedure alternates between which element is the left or right element in the quasigroup operation, where $Y_1 = X_1$. Finally, the

string $\mathbf{y} = Y = Y_1 Y_2 \cdots Y_k$ is created as the final output, seen as bit string of length $n = 8 \cdot k$. Algorithm 4.2 gives the full details of the method.

By the definition of a quasigroup, the central map $\mathcal{P}'$ is guaranteed to be an invertible transformation. Because the first vector X_1 is mapped directly to its output Y_1, it is possible to "unwrap" the chain of quasigroup operations applied in Figure 4.2 by using the appropriate division operator. We will see how this is done in the next section.

Since the quasigroup is an MQQ, the central map $\mathcal{P}'$ will be a quadratic transformation. Composed with the other two functions, $\mathcal{S}$ and $\mathcal{S}'$, we get a public key $\mathcal{P}$ consisting of n quadratic multivariate polynomials.

Algorithm 4.2 The MQQ-SIG central map $\mathcal{P}'$

Input: A vector $\mathbf{x} \in \mathbb{F}_2^n$, where n is required to be a multiple of 8. Additionally it assumed that an MQQ, $(Q, *)$, is already defined.
Output: A vector $\mathbf{y} \in \mathbb{F}_2^n$.

1. Regard the vector $\mathbf{x}$ as a string $X = X_1 X_2 \ldots X_k$, where $k = \frac{n}{8}$, and each X_i is an 8-element vector representing an element in Q;
2. Compute the string $\mathbf{y} = Y_1 Y_2 \ldots Y_k$, where

$$Y_i = \begin{cases} X_1 & i = 1 \\ X_{i-1} * X_i & i = 2, 4, \ldots k \\ X_i * X_{i-1} & i = 3, 5, \ldots k-1 \end{cases} \tag{4.7}$$

3. Let $\mathbf{y}$ be the final output, regarded as an element in $\mathbb{F}_2^n$.

4.2.3 The inverse map $\mathcal{P}'^{-1}$

Having dealt with the forward direction of the central map $\mathcal{P}'$, we now turn to its inverse. Let $\mathbf{y}$ denote the n-bit input vector coming from the $\mathcal{S}^{-1}$ function (cf. Equation 4.3). As in the case of the forward direction, the input is split into $\frac{n}{8}$ vectors, $Y_i \in \mathbb{F}_2^8$. Each of these vectors can be considered elements in the MQQ. To invert the $\mathcal{P}'$ function we seek to reverse the action done in Equation 4.7. To this end, the values X_i are obtained by using the quasigroups left and right division

operators as follows:

$$X_i = \begin{cases} Y_1 & i = 1 \\ X_{i-1} \backslash Y_i & i = 2, 4, \ldots, k \\ Y_i / X_{i-1} & i = 3, 5, \ldots, k-1 \end{cases}$$

The process is also illustrated in Fig. 4.3. This unwrapping of the X_i values explains why it was necessary to set $Y_1 = X_1$ in Equation 4.7.

After computing all the left and right divisors, the preimage $\mathbf{x} = \mathcal{P}'^{-1}(\mathbf{y})$ is obtained by forming the bit string $\mathbf{x} = X_1 X_2 \ldots X_k$.

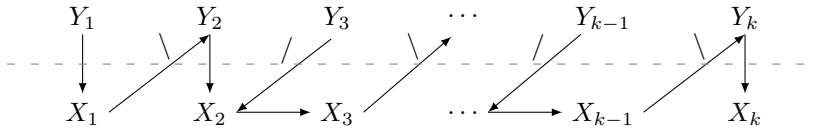


Figure 4.3: A graphical presentation of the construction of the inverse map $\mathcal{P}'^{-1}$, using the right and left division operators.

So far we have not said anything about how to actually compute the division operators $\backslash$ and $/$. Unlike the $*$ operator, we do not have a closed form expression for $\backslash$ and $/$ readily available. In general, obtaining a closed form formula can be computationally demanding, since the division operators may not be quadratic. In fact, the division operators can be of any degree deg where $2 \leq deg \leq d$ [SCG12]. Consequently, the division operators are usually implemented in a way that does not require a closed form formula. We present two different approaches below.

The simplest approach is to pre-compute the multiplication tables for each operation based on the original operator $*$. Then $\backslash$ and $/$ can be implemented as efficient table lookups. Since the quasigroups in MQQ-SIG are of size 2^8, this requires two tables of size 64 KB to be stored. For anything larger than 2^8, this approach quickly becomes prohibitively expensive, so it limits the MQQ's to be of relatively small sizes.

Another approach, which avoids the storage requirements of the division tables, is to invert $\mathcal{P}$ by solving a system of d linear equations. Recall the following relation between $*$ and the division operators:

$$\mathbf{x} * \mathbf{y} = \mathbf{z} \Leftrightarrow \mathbf{x} = \mathbf{y} \backslash \mathbf{z} \Leftrightarrow \mathbf{y} = \mathbf{z} / \mathbf{x}.$$

Given two known values $\mathbf{x}, \mathbf{z} \in \mathbb{F}_2^8$. To find $\mathbf{y} = \mathbf{z}/\mathbf{x}$, say, we need to solve the following system of equations:

$$\mathbf{z} = \mathbf{x} * \mathbf{y}.$$

Since the quasigroups in MQQ-SIG are bilinear, this amounts to solving a system of linear equations in the variables $y_1, y_2, \ldots, y_d$. Using Gaussian elimination this can be done in approximately $O(d^3)$ time. This method reduce the storage requirements at the expense of somewhat longer run times. Note that this method for inverting the central map is similar to the approach used in UOV (cf. Section 2.3).

4.3 The transformations $\mathcal{S}$ and $\mathcal{S}'$

The two transformations $\mathcal{S}$ and $\mathcal{S}'$, which constitute the linear and affine part of the public key in MQQ-SIG, are determined by an invertible $n \times n$ Boolean matrix S. In general, S could be generated uniformly at random, requiring n^2 bits to store. With the proposed sizes of $n = 160, 192, 224$ or 256, this amounts to an addition of 3 to 8 KBytes to the private key. In order to reduce the size of the private key, the designers of MQQ-SIG suggested to create S in a special manner, using the concept of *circulant matrices*. A circulant matrix is a square matrix, where each row vector is rotated one element to the right relative to the preceding row vector.

The invertible matrix S is defined in terms of its inverse S^{-1}, which is created as the sum of two circulant matrices as follows:

$$S^{-1} = \bigoplus_{i=0}^{\frac{n}{16}} I_{\sigma_i^0} \oplus \bigoplus_{i=0}^{\frac{n}{16}+1} I_{\sigma_i^1}, \tag{4.8}$$

where $I_{\sigma_i^0}$, $i = \{0, 1, 2, \ldots, \frac{n}{16}\}$ and $I_{\sigma_i^1}$, $i = \{0, 1, 2, \ldots, \frac{n}{16} + 1\}$ are permutation matrices of size n, the operation $\oplus$ is a bitwise exclusive-OR (XOR) of the elements in the permutation matrices, and σ_i^0, σ_i^1 are permutations on n elements. They are defined by the following expressions:

$$\begin{cases} \sigma_0^0 - \text{ random permutation on } \{1, 2, \ldots n\}, \\ \sigma_i^0 = \text{RotateLeft}(\sigma_{i-1}^0, 8), \text{ for } i = \{1, \ldots, \dfrac{n}{16}\}, \\ \sigma_0^1 - \text{random permutation on } \{1, 2, \ldots n\}, \\ \sigma_i^1 = \text{RotateLeft}(\sigma_{i-1}^1, 8), \text{ for } i = \{1, \ldots, \dfrac{n}{16} + 1\}. \end{cases}$$

Of course, σ_0^0 and σ_0^1 must be chosen such that Equation 4.8 yields an invertible matrix S^{-1} (and hence also $S = (S^{-1})^{-1}$).

From S we define $\mathcal{S}$ to be the linear transformation

$$\mathcal{S}(\mathbf{x}) = S \cdot \mathbf{x}, \tag{4.9}$$

and $\mathcal{S}'$ to be the affine transformation

$$\mathcal{S}'(\mathbf{x}) = S \cdot \mathbf{x} + \mathbf{v}. \tag{4.10}$$

The vector $\mathbf{v} = (v_1, v_2, \ldots, v_n) \in \mathbb{F}_2^n$ is derived from the permutation $\sigma_0^1 = \left(\begin{smallmatrix} 1 & 2 & \cdots & n \\ s_1 & s_2 & \cdots & s_n \end{smallmatrix}\right)$ in the following peculiar manner:

$$v_i = \left(\left(\frac{\left(\left(s_{1+\lfloor \frac{i-1}{8} \rfloor} \right) \bmod 16 \right) \times 16}{2^{(8-i) \bmod 8}} \right) + \left(\frac{s_{65+\lfloor \frac{i-1}{8} \rfloor}}{2^{(8-i) \bmod 8}} \right) \right) \bmod 2. \tag{4.11}$$

Put less succinctly: the bits v_i, of the vector $\mathbf{v}$, are constructed as the XOR of two terms. The first term is the four least significant bits from one of the values $s_1, \ldots, s_{\frac{n}{8}}$, shifted four places to the left, and then shifted between 0-7 places to the right again, depending on the value of i. The second term is one of the values $s_{65}, \ldots, s_{65+\frac{n}{8}}$ shifted the same variable number of places to the right.

Gligoroski et al. [GØJ+11], explains the construction of S both from a performance and a security perspective. As mentioned in the beginning of the section, the goal was to reduce the private key size. Since S^{-1} is uniquely determined by σ_0^0 and σ_0^1, it is not necessary to store the full matrix. We will come back to this point in the next section.

The security argument regards the matrix' properties compared to a truly random matrix. The matrix is constructed as the combination of two circulant matrices instead of using just one circulant matrix. Since the inverse of a circulant matrix is again circulant, which has strong regular properties, they were afraid that it might affect the randomness of the multivariate quadratic equations in the public key. Thus, they chose to use two permutations instead of just one to avoid having a circulant matrix as one of the transformation in the public key. The cost is in the need to store two permutations instead of one.

4.4 Operating characteristics

4.4.1 Sizes of the public and private key

The public key in MQQ-SIG consists of $\frac{n}{2}$ randomly generated multivariate quadratic equations. From Equation 2.2, we know that each polynomial might have $1 + \frac{n(n+1)}{2}$ terms. Since we need one bit to store each potential term, the size of the public key is $\frac{n}{2} \times (1 + \frac{n(n+1)}{2})$ bits. For MQQ-SIG the number of variables are $n \in \{160, 192, 224, 256\}$, which yields a public key in the range of 128 to 412 KB.

The private key is the triple $(\mathcal{S}^{-1}, \mathcal{P'}^{-1}, \mathcal{S'}^{-1})$, but in Algorithm 4.1 it is not specified which parts of the key that need to be stored, because this can be implementation specific. As explained in Section 4.2.3, the performance of the division operators can be seen as a time-memory trade-off between faster run times, at the expense of having to store two lookup tables; and use less memory, at the cost of having to solve a set of linear equations.

These considerations can be applied to the transformations $\mathcal{S}^{-1}$ and $\mathcal{S'}^{-1}$ as well. They are both defined in terms of the matrix S^{-1} and the vector $\mathbf{v}$. Recall that S^{-1} is created in a special way using two circular matrices defined by the permutations σ_0^0 and σ_0^1. Since σ_0^0 and σ_0^1 uniquely determine S^{-1} (and $\mathbf{v}$), it is sufficient to just store them as the representation of the private-key transformations $\mathcal{S}^{-1}$ and $\mathcal{S'}^{-1}$. Consequently, we have the following result:

Proposition 4.4. *The linear transformation S^{-1} can be encoded in a unique way using $2n$ bytes.*

Proof. Since σ_0^0 and σ_0^1 are sufficient to uniquely recreate S^{-1} it is only necessary to find an encoding of these two permutations. A permutation on n symbols can be uniquely encoded as a string of n bytes, where each byte represents one of the symbols, so the claim follows. □

For a system which implements the signing part of MQQ-SIG it is possible to either just store the permutations σ_0^0 and σ_0^1, and then recreate S^{-1} for each signature, or to store the full matrix S^{-1} (and $\mathbf{v}$), saving a few cycles when generating a signature. One might be preferable over the other depending on context.

Finally, the quasigroup needs to be stored, which in MQQ-SIG requires 81 bytes.

Proposition 4.5. *A multivariate quadratic quasigroup created according to Equation 4.1, with $d = 8$, can be encoded in unique way with 81 bytes.*

Proof. Recall that each element in the MQQ vector is a bilinear function of the form

$$\sum_{1 \leq i,j \leq d} a_{ij} x_i y_j + \sum_{i=1}^{d} b'_i x_i + \sum_{i=1}^{d} b''_i y_i + c,$$

where $a_{ij}, b'_i, b''_i, c \in \mathbb{F}_2$. For $d = 8$ the quadratic part can potentially require $64 = \frac{8^2}{2}$ bits to store, the linear part $16 = 2 \times 8$ bits, and the constant 1 bit. Since we have 8 such functions the total storage requirement is $648 = 8 \times (64 + 16 + 1)$ bits, or 81 bytes. □

To summarize: the size of the private key in MQQ-SIG can be as low as $401 = 81 + 2 \times 160$ bytes, if only the permutations and the quasigroup are stored, or as high as 136 KB if the division operators are implemented as pre-computed lookup tables and the full matrix S is stored.

4.4.2 Computational aspects

The designers of MQQ-SIG also provided some performance results of the system, based on a C implementation submitted to the SUPERCOP benchmarking system [BL]. We restate their results here in Table 4.1 for easy reference. The test also included performance numbers from other signature schemes, including RSA [RSA78] and ECDSA [JMV01], in addition to several $\mathcal{MQ}$-schemes: Rainbow [DS05], TTS [YC05] and 3ICP [DWY07].

The results in Table 4.1 show that MQQ-SIG is much faster than both RSA (600-640 times) and ECDSA (270-360 times) when it comes to signing a message of 59 bytes. It also performs significantly better than the other $\mathcal{MQ}$-schemes, in the range from 10 to 400 times faster.

The verification speed is close to that of RSA and ECDSA, and somewhat slower than the $\mathcal{MQ}$-schemes. This was explained to be due to non-optimized code for this part of the algorithm, and that they except to see results comparable to the other $\mathcal{MQ}$-schemes with a better implementation. This is reasonable since the verification in MQQ-SIG follows a relatively standard approach, found in most $\mathcal{MQ}$-systems.

The worst performing part of MQQ-SIG is the key generation algorithm. Again, this is explained to be partially due to unoptimized code. Still, the designers admit that the key generation will probably be the most time consuming part of the system.

Table 4.1: Comparison between RSA, ECDSA, and several $\mathcal{MQ}$ schemes: MQQ-SIG, Rainbow, TTS and 3ICP. Operations have been performed in 64-bit mode of operation on an Intel Core i7 920X machine running at 2 GHz. Table adopted from [GØJ+11].

Security level	Algorithm	KeyGen (CPU cycles)	Signing of 59 bytes (CPU cycles)	Verification of a signature on 59 bytes (CPU cycles)	Private key size (bytes)	Public key size (bytes)
2^{80}	RSA1024	102,869,553	2,213,1112	60,084	1024	128
	ECDSA160	1,201,188	944,364	1,083,060	60	40
	MQQSIG160	1,509,486,980	3,476	97,740	401	137,408
	RainbowBinary256	30,311,648	38,784	43,800	23,408	30,240
2^{96}	RSA1536	322,324,271	5,452,076	87,516	1536	192
	ECDSA192	1,799,284	1,390,560	1,662,664	72	48
	MQQSIG192	1,894,161,224	4,264	72,652	465	222,360
2^{112}	RSA2048	786,466,598	11,020,696	125,776	2048	256
	ECDSA224	2,022,896	1,555,740	1,821,348	84	56
	MQQSIG224	3,140,892,292	4,756	95,180	529	352,828
2^{128}	RSA3072	2,719,353,538	31,941,760	230,536	3072	384
	ECDSA256	2,296,976	1,780,524	2,085,588	96	64
	MQQSIG256	4,839,469,440	4,948	138,324	539	526,368
	TTS6440	60,827,704	84,892	76,224	16,608	57,600
	3ICP	15,520,100	1,641,032	60,856	12,768	35,712

CHAPTER 5

Experimental Analysis

The notion of *weak keys* is an important concept in cryptography. Let $\mathcal{C}$ be a cryptographic scheme and k a key in its key space K. We say that k is a weak key for $\mathcal{C}$, if it has some properties that cause the cipher to behave in some undesirable way; normally by making cryptanalysis easier. The opposite notion of weak keys, is *strong keys*. While it would be preferable to only use strong keys, the designers of a cipher is usually more concerned with avoiding weak keys in the system. Weak keys can be found in both symmetric and asymmetric cryptography, but is especially common in public-key schemes. This is because they often are based on a highly algebraic formulation, which makes them more susceptible to contain keys that deviate from the norm due to some special mathematical properties.

To illustrate the concept of weak keys, consider the selection of primes used in RSA. There are several factorization algorithms that are highly efficient when p and q are of some special form. For instance, the "Pollard $p-1$ attack" works very well if the factors of $p-1$ (or $q-1$) contains many small prime factors; and the "Fermat factorization method" can be efficient if p and q are not too far apart. To thwart such attacks, special care should be taken when choosing the primes. In particular, they should not be too close together and be of a special form called

strong primes[*]. Although, the purported benefit of this added complexity has been disputed by Rivest and Silverman [RS99], it goes to show that choosing the keys in a cryptosystem is not trivial matter. Furthermore, we have not even considered attacks other than factorization for breaking RSA, such as Wiener's attack on low decryption exponents [Wie90] and Coppersmith's attack on low encryption exponents [Cop97].

The notion of weak and strong keys can be extended to also include parameters that are not technically keys in the cryptographic scheme, but are important for its behavior nevertheless. An example is elliptic curve cryptography, which is based on the discrete logarithm problem (DL) in special groups generated by the points on an elliptic curve. Shoup showed that there is a lower bound on how efficient generic DL-algorithms can be [Sho97b]. That is, when no special properties of the group are used to solve the problem, finding discrete logarithms is a hard problem. Nevertheless, there is a special class of elliptic curves that yields groups in which a related problem, called the *Decisional Diffie-Hellman* problem (DDH)[†], is easy solvable. These elliptic curves are called *supersingular*, and represent a class of public parameters that cryptosystems based on elliptic curves should avoid.

In this chapter we try to identify different classes of keys found in the MQQ-SIG scheme. We do so by performing extensive numerical analysis on the various parameters that defines the system. So far, little is known about what each parameter actually contributes to the security of the system. By our analysis we hope to shed some light on the role each parameter plays in the security of MQQ-SIG and to identify choices of parameters that lead to weak keys. In the next chapter we propose a modified key generation algorithm that accounts for the findings obtained here.

The original public-key cryptosystem based on multivariate quadratic quasi-groups ([GMK08a]) was broken using different variations of Gröbner basis analysis [MDBW09]. Computing Gröbner bases is a general-purpose method for solving systems of multivariate equations. We provide a brief introduction on the basics of Gröbner bases, and the algorithms used to compute them, in the next section.

In addition to the general method of directly solving the system of polynomial

---

[*]A prime p is strong if it is of the form $p = aq_1 + 1 = bq_2 - 1$, where both q_1 and q_2 are large prime numbers.

[†]See for instance Chapter 8 and 13 in [Mao03].

equations (using Gröbner bases or otherwise), there are other attacks that seek to exploit the internal algebraic structure of $\mathcal{MQ}$-schemes. These specialized attacks can be much more effective than generic attacks since they incorporate more knowledge about the internals of the problem. In the cryptanalysis literature there is an abundance of broken $\mathcal{MQ}$-schemes due to attacks other than system solving. Of course, these attacks are usually less applicable to other ciphers since they are intimately related to the specifications of a particular system. Nevertheless, such attacks can be devastating for the cryptographic scheme. Currently there are no known attacks that specifically target the inherent algebraic properties of MQQ-SIG. In this thesis we only consider weaknesses against Gröbner bases attacks in our analysis.

5.1 Gröbner basis cryptanalysis

The concept of a Gröbner basis was developed in the sixties by B. Buchberger. It has applications in many different areas of mathematics, such as algebraic geometry, number theory, homological algebra and combinatorics. For our purposes, their usefulness is as a tool for solving systems of multivariate polynomial equations. Let K be a field and $P = K[x_1, \ldots, x_n]$ the ring of multivariate polynomials. Given a system of polynomial equations:

$$f_1(x_1, \ldots, x_n) = \ldots = f_r(x_1, \ldots, x_n) = 0, \tag{5.1}$$

we call the common set of zeroes for $f_1, \ldots, f_r$ the *algebraic variety* of the system. Gröbner bases allow us to describe the algebraic variety for a set of polynomial equations in a "nice" manner. What makes Gröbner bases so special is that they are *computable*. We will later indicate how this can be done. First, we proceed with a definition of a Gröbner basis. In following we will state some theorems without proofs, we refer to [KR08] for details and further information on the theory of Gröbner bases.

Let $I = \langle f_1, f_2, \ldots, f_r \rangle$ be an ideal in P, generated by the polynomials in the set $\{f_1, f_2, \ldots, f_r\}$. We call this set a *basis* for the ideal I. Unlike the situation in linear algebra, there is no requirement of independence between the polynomials in order to form a basis. It can be easily shown that the algebraic variety of the

polynomials $f_1, f_2, \ldots, f_r$ is the same as their generated ideal. Given a basis for an ideal I in P we would like to modify it, if possible, to better describe the variety of the ideal.

To this end, the notion of a *division algorithm* is an important tool for the task. For the single variable case $K[x]$, we know that there exists a division algorithm such that for two polynomials $f, g \in K[x]$, there exists unique polynomials q, r in $K[x]$ such that $f(x) = q(x)g(x) + r(x)$. Furthermore, either $r(x) = 0$ or the degree of $r(x)$ is less than the degree of $f(x)$. For the multivariate case there exists an analogous algorithm, but with some important differences from the univariate case. We state some of its properties in the following theorem. For ease of notation we denote $K[x_1, x_2, \ldots, x_n]$ by $K[\mathbf{x}]$ and a polynomial f in $K[\mathbf{x}]$ as $f(\mathbf{x})$.

Theorem 5.1. *Let f, g, q and r be polynomials in $K[\mathbf{x}]$ such that $f(\mathbf{x}) = q(\mathbf{x})g(\mathbf{x}) + r(\mathbf{x})$. The common zeroes of f and g are the same as the common zeroes of g and r. Also, if f and g are two members of a basis for an ideal I of $K[\mathbf{x}]$, then replacement of f by r in the basis still yields a basis for I.* □

A natural goal in trying to find a nice basis for an ideal I in $K[\mathbf{x}]$, is to either replace the polynomials in the basis with polynomials of lower degree, or to reduce the number of variables. Theorem 5.1 says that this can safely be done, when possible.

Example 5.2. Let $\{xy^2, y^2 - y\}$ be a basis for the ideal $I = \langle xy^2, y^2 - y\rangle$ in $\mathbb{R}[x, y]$. We see that y^2 divides xy^2, so we carry out the division algorithm to obtain the remainder of xy^2 divided by $y^2 - y$:

$$\begin{array}{r|l} & x \\ \hline y^2 - y & xy^2 \\ & xy^2 - xy \\ \hline & xy \end{array}$$

By Theorem 5.1 we can now replace xy^2 in the basis with xy. This gives us the new basis $\{xy, y^2 - y\}$ for I where one of the polynomials has a lower degree then in the original basis. Since y^2 does not divide xy we cannot continue the process. △

In order to conduct this reduction in a systematic step-by-step manner, the notion of *term ordering* is necessary. For any multivariate polynomial there can be imposed an order upon its terms based on a *monomial ordering*. Recall that a monomial (or power product) is an element in $K[\mathbf{x}]$ of the form

$$x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n},$$

where all the $\alpha_i \geq 0$ in $\mathbb{Z}$. We call the sum $\sum_{i=1}^{n} \alpha_i$, the *total degree* of the monomial.

Definition 5.3. A *monomial ordering* upon $K[\mathbf{x}]$ is a total ordering $<$ such that the following conditions are satisfied for all monomials $m, m_1, m_2 \in K[\mathbf{x}]$.

1. $1 < m$, if $m \neq 1$.
2. If $m_1 < m_2$, then $mm_1 < mm_2$.
3. The ordering is well-ordered (every non-empty subset of monomials in $K[\mathbf{x}]$ have a minimal element with respect to $<$).

The ordering of multivariate monomials is not unique, however; it can be defined in several different ways. To illustrate, we provide the definition of two common term orderings; namely the lexicographical order and the graded reverse lexicographical order.

Definition 5.4. Let $t_1 = ax_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$ and $t_2 = bx_1^{\beta_1} x_2^{\beta_2} \cdots x_n^{\beta_n}$ be two terms in $K[\mathbf{x}]$. Then $t_1 <_{\texttt{Lex}} t_2$ if and only if $\alpha_i < \beta_i$ for the first subscript i, reading left to right, such that $\alpha_i \neq \beta_i$. We call this ordering the *lexicographical term order*.

Definition 5.5. Let $t_1 = ax_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$ and $t_2 = bx_1^{\beta_1} x_2^{\beta_2} \cdots x_n^{\beta_n}$ be two terms in $K[\mathbf{x}]$. Then $t_1 <_{\texttt{Grevlex}} t_2$ if and only if the total degree of t_1 is less than the total degree of t_2; or if the total degree is equal, t_1 is less than t_2 with respect to the lexicographical order applied to the variables in reverse order. We call this ordering the *graded reverse lexicographical order*.

Example 5.6. Given the four terms $x_1x_2^2x_3$, x_3^2, x_1^3 and $x_1^2x_3^2$ in $K[x_1, x_2, x_3]$, then their lexicographical ordering is

$$x_3^2 <_{\texttt{Lex}} x_1x_2^2x_3 <_{\texttt{Lex}} x_1^2x_3^2 <_{\texttt{Lex}} x_1^3,$$

and their graded reverse lexicographical ordering is

$$x_3^2 <_{\texttt{Grevlex}} x_1^3 <_{\texttt{Grevlex}} x_1^2 x_3^2 <_{\texttt{Grevlex}} x_1 x_2^2 x_3.$$

△

In the following we will consider every polynomial in $K[\mathbf{x}]$ to be written in decreasing order of terms with respect to some term ordering; that is, the first term has the highest order. We denote by $\mathrm{LT}(f)$ the first term of a polynomial f with respect to the term ordering, and call it the *leading term* of f. Looking back at Example 5.2, we see that the new basis we obtained is also of smaller maximal lexicographical term size than the original basis. Note however, that $\mathrm{LT}(xy) \not<_{\texttt{Lex}} \mathrm{LT}(y^2 - y) = y^2$.

At last we define what it means for a set of polynomials to be a Gröbner basis for an ideal I.

Definition 5.7. A set $G = \{f_1, \ldots, f_n\}$ of non-zero polynomials in $K[x_1, \ldots, x_n]$, with term ordering $<$, is a *Gröbner basis* for the ideal $I = \langle f_1, \ldots, f_n \rangle$ if for all $f \in I$, there exists some $g \in G$ such that $\mathrm{LT}(g)$ divides $\mathrm{LT}(f)$.

The following theorem tells us how Gröbner bases can be used to solve a system of multivariate polynomials over $\mathbb{F}_2$.

Theorem 5.8. *The Gröbner basis of the system* $\{f_1, \ldots, f_n, x_1^2 - x_1, \ldots, x_n^2 - x_n\}$ *in* $\mathbb{F}_2[x_1, \ldots, x_n]$*, is of the form* $G = \{x_1 - a_1, \ldots, x_n - a_n\}$*, where* $a_i \in \mathbb{F}_2$*. Furthermore,* $(a_1, \ldots, a_n)$ *is exactly the solution in* $\mathbb{F}_2$ *to the system of equations.* □

The original method for computing Gröbner bases was Buchberger's algorithm, introduced in his PhD-thesis [Buc65]. The basic idea of the algorithm is the following: continuously pick out two polynomials g_i, g_j, from the original basis and multiply their leading terms with a monomial as small as possible, such that their product becomes the same (called their *least common multiple*). Add them together with suitable coefficients from K so that cancellation of their leading terms ensues. Reduce the resulting polynomial, often denoted $S(g_i, g_j)$, relative to the rest of the polynomials in the basis using the multivariate division algorithm. If $S(g_i, g_j)$ can be reduced to zero, pick another pair of polynomials in the basis and repeat the

process. If $S(g_i, g_j)$ cannot be reduced to zero, add it to the basis and start over, reducing this basis as much as possible.

Buchberger proved that the above process will terminate. In particular, a Gröbner basis is found when every polynomial $S(g_i, g_j)$ for all $i \neq j$ can be reduced to zero using polynomials from the most current basis. This is due to the following important criterion for determining whether a basis constitute a Gröbner basis.

Theorem 5.9. *A basis $G = \{g_1, g_2, \ldots, g_r\}$ is a Gröbner basis for the ideal $I = \langle g_1, g_2, \ldots, g_r \rangle$ if and only if, for all $i \neq j$, the polynomial $S(g_i, g_j)$ can be reduced to zero by repeatedly dividing remainders by elements of G.* □

Although the procedure will eventually finish, it may take a very long time. In the worst-case the run time can be as bad as doubly exponential, that is, $O(2^{2^n})$ in the number of variables n [BFS03]. Furthermore, the intermediary bases which are computed in the process can be very large, giving high memory requirements.

In 1999 Faugère proposed a much improved Gröbner basis algorithm called F_4 [Fau99], and later a refinement called F_5 [Fau02]. These algorithms build on the Buchberger algorithm, but enhance it by reducing the problem to a sparse linear algebra problem instead. More precisely, F_4/F_5 incrementally construct matrices A_D of degree $2, 3, \ldots, D$; each row containing polynomials of the form $m_j f_{i_j}$, where $\{f_{i_j}\}$ is some subset of the polynomials from the current basis and $m_1, m_2, \ldots$ are monomials such that the total degree of $m_j f_{i_j}$ is less than D. Then the row echelon form of the A_D matrices are computed using linear algebra techniques.

The number D represents the maximal total monomial degree that occurs during the computation. This parameter is called the *degree of regularity* and is a key factor in determining the complexity of Gröbner basis algorithms. For the F_4/F_5 algorithms in particular, it gives the size n^D of the matrices A_D, where n is the number of variables. This yields an overall complexity of

$$O(n^{\omega D}),$$

where $2 < \omega \leq 3$ is the "linear algebra constant". For a random system of multivariate equations, the asymptotic behaviour of the maximal degree occurring in the computation is on the order $D \approx \frac{n}{11.114\ldots}$, see [FJ03, BFS03].

In addition to the F_4/F_5 algorithms, another method for solving systems of multivariate quadratic equations called XL, was proposed by Courtois et al. [CKPS00].

However, it has been shown (see [ACFP12]) that XL, and its descendant family of "mutant" algorithms called MXL [MMDB08], are actually Gröbner basis algorithms as well. Furthermore, they can be considered redundant versions of the F_4 algorithm.

Both the F_4 algorithm and MutantXL have been used to break the original MQQ cipher [MDBW09]. In a subsequent cryptanalysis of the cipher (see [FØPG10]), it was shown that the reason for the MQQ cryptosystem to be so weak against Gröbner basis attacks was that its degree of regularity was bounded from above by a very small constant. Hence, even if more variables are added to the system it will not increase the complexity in any substantial manner.

5.2 The parameters of MQQ-SIG

In order to be able to separate good keys from bad keys in the key generation, we need to understand all the parameters that defines the key. In this section we discuss all the constituent parts of the public key in MQQ-SIG. We identify their role in the algorithm, and how their properties might affect the strength of the keys. In the following sections we will present an extensive experimental study on each of the factors identified here.

Recall from Chapter 4 that the public key $\mathcal{P}$ in MQQ-SIG is created as the composition of three transformations:

$$\mathcal{P} = \mathcal{S} \circ \mathcal{P}' \circ \mathcal{S}',$$

where $\mathcal{S}$ and $\mathcal{S}'$ are linear and affine transformations, respectively, while the central quadratic map $\mathcal{P}'$ is the application of the quasigroup operation. The specific procedure for how to apply the quasigroup operation is an intrinsic part in the design of MQQ-SIG and not actually relevant for its key generation. Hence, it will not be considered further here. Additionally, $\mathcal{S}$ and $\mathcal{S}'$ are completely determined by the matrix S, so in reality there are only two main factors to consider in MQQ-SIG; the matrix S and the quasigroup itself. First we look at the quasigroup construction.

The quasigroup is constructed according to Equation 4.3 and has the following parameters: three random invertible Boolean matrices A_1, A_2 and B, an upper triangular matrix $U(\mathbf{x})$ of linear expressions in $\mathbf{x}$, and a Boolean vector $\mathbf{c}$. Disregarding

A_1, A_2, B and $\mathbf{c}$ for the moment, we get the expression

$$\mathbf{x} * \mathbf{y} = U(\mathbf{x}) \cdot \mathbf{y} + \mathbf{x}, \tag{5.2}$$

which is a multivariate quadratic quasigroup according to [CKG10]. We see immediately that $U(\mathbf{x})$ is a very central parameter in the quasigroup construction. More specifically, $U(\mathbf{x})$ defines an isotopy class on which A_1, A_2 and B all give linearly isotopic quasigroups within the same equivalence class. The concept of isotopy classes lead naturally to some questions regarding their security properties.

1. The quasigroup established in Equation 5.2 is rather simple. Many properties are shared between quasigroups in the same isotopy class. How will the isotopies created by A_1, A_2 and B influence the behavior of the system? We look at A_1, A_2 and B separately in Section 5.6 and Section 5.7, in order to answer these questions.

2. Building upon the previous observation, it is also prudent to ask if there exists isotopy classes which have inherently better or worse properties than others. This question will be studied in Section 5.5.

Lastly, we look at the matrix S. The standard way to hide the inner structure of the central map in most $\mathcal{MQ}$ schemes, is to use two random affine/linear transformations $\mathcal{S}$ and $\mathcal{T}$. In MQQ-SIG, these transformations are determined by the specially constructed matrix S. How will this choice affect the security of the system? Is it better to follow the standard approach? We investigate these questions in Section 5.8.

In the sections to follow, we hope to be able to distinguish good keys from bad keys, using extensive numerical analysis. To more easily pinpoint the exact feature of a parameter that leads to a strong or weak key, we have chosen to study each parameter separately. That is not to say that we believe the combination of the components are unimportant. To the contrary, given the complexity of the Gröbner basis algorithms and how little is known about what makes them efficient, it should be conceivable that the combination of different parameters might lead to different results than when varying only one parameter separately. Nevertheless, for ease of analyzing,we have regarded it as outside the scope of this thesis.

5.3 Experimental procedure

5.3.1 Hardware and software

For the experiments we used a 64-bit Ubuntu Linux server, version 10.04.4/2.6.32, with 64 Intel Xeon 2.27 GHz processors and 1 TB of RAM. To run the attacks we used the computer algebra system Magma [BCFS], (version 2.17-3). Magma implements several algorithms for computing Gröbner bases, including the original algorithm by Buchberger [Buc65] and the new (and faster) F_4-algorithm due to Faugère [Fau99]. For the special case of multivariate equations over $\mathbb{F}_2$, F_4 is the algorithm of choice.

The Gröbner basis function in Magma generates a considerable amount of output on each run. It provides much information on how its run time characteristics changes from each stage of the algorithm. Of particular interest to us is the total time needed to compute the basis and the degree of regularity occurring during the run of the algorithm. Memory consumption is of course also an important factor to consider, but with 1 TB of RAM available, it was not a major concern for us.

5.3.2 Attack algorithm

For each experiment the attack algorithm was the same, and is specified in Algorithm 5.1. Notice that the system $\widetilde{\mathcal{P}}$, given to the Gröbner basis algorithm in Step. 6 is not an instance of a "pure" MQQ-SIG public key. Recall that in MQQ-SIG half of the equations in the public key are discarded. Thus, for a real MQQ-SIG public key, the system is actually underdetermined ($n = 2m$). To solve an underdetermined system of multivariate quadratic equations, Courtois et. al [CGM$^+$02] proposed to "fix" some of the n variables (e.g. simply by choosing their values at random) to obtain a system with $n \leq m$. The resulting system can then be solved using an algorithm which is efficient at solving (over)-determined systems, for instance the XL algorithm [CKPS00].

In our attack algorithm we simulate this approach by evaluating r of the variables in the public key at random values (Step. 4).

Algorithm 5.1 Gröbner basis attack used for all experiments

Input: r; the number of equations to remove from the public key.
Output: A Gröbner basis for the public key.

1. Generate the public key $\mathcal{P} = \mathcal{S} \circ \mathcal{P}' \circ \mathcal{S}'$ according to the specification of the experiment.
2. Remove the first r equations of $\mathcal{P}$.
3. Generate a random Boolean vector $\mathbf{x} = (x_1, x_2, \ldots, x_r) \in \{0,1\}^r$.
4. Evaluate each of the remaining $n-r$ polynomials $P_i(x_1, \ldots, x_n)$, at the random vector $\mathbf{x}$, for $i = r+1, \ldots, n$.
5. Obtain a system $\widetilde{\mathcal{P}} = \{P_i(x_{r+1}, \ldots, x_n) | i = r+1, \ldots, n\}$ of $n-r$ equations in $n-r$ variables.
6. Call the function GroebnerBasis($\widetilde{\mathcal{P}}$) from Magma to find a Gröbner basis for the system $\widetilde{\mathcal{P}}$.

5.4 Experiments on the original MQQ-SIG

To provide a benchmark for further experiments, we conducted several measurements on the time to compute the Gröbner basis for an unmodified MQQ-SIG system. In the following we will refer back to these benchmarks as the "original experiments", and the generated public keys as the "original instances".

Due to the complexity of these experiments (with respect to run time and memory requirements), we only ran the MQQ-SIG scheme with 32, 40, 48 and 56 variables. In essence, our procedure can be summarized as follows:

1. generate 100 random MQQ-SIG public keys for each number of variables, following the description in Chapter 4[‡];
2. for each instance, store all parameters necessary to faithfully restore it later;

[‡]We had to make one adjustment from the original specification. In the definition of the Boolean vector $\mathbf{v}$ (Equation 4.11) it is assumed that the number of variables n in the system will be 160 or more. Hence it is guaranteed that the expression $s_{65+\lfloor \frac{i-1}{8} \rfloor}$, will always be well-defined. In our experiments $n \leq 56$, so the previous value will not be valid. To circumvent this problem we instead defined this value as: $s_{\lfloor \frac{2n}{3}+1 \rfloor + \lfloor \frac{i-1}{8} \rfloor}$. We believe that this change will not have affected the security of the system in any significant way.

3. run Algorithm 5.1 with $8 \leq r \leq \frac{n}{2}$ on each public key.

The run times, averaged over all the instances of a given number of variables, are presented both in Figure 5.1 and Table 5.1. The graph clearly shows the exponential nature of adding more equations and variables to the system. What is more interesting to note, is the very large standard deviation shown in Table 5.1. Among the 100 test instances of each number of variables, there were some instances that performed a little better than average, but nothing significantly. Several instances performed considerably worse than average. This is a strong indication of the existence of weak keys in the MQQ-SIG scheme. To better illustrate the prevalence of these bad instances, we provide the full graph in the case $n = 48$ in Figure 5.2. As this plot shows, a few instances are very bad. The goal of the next sections, will be to the establish the exact properties of the keys that results in this behavior.

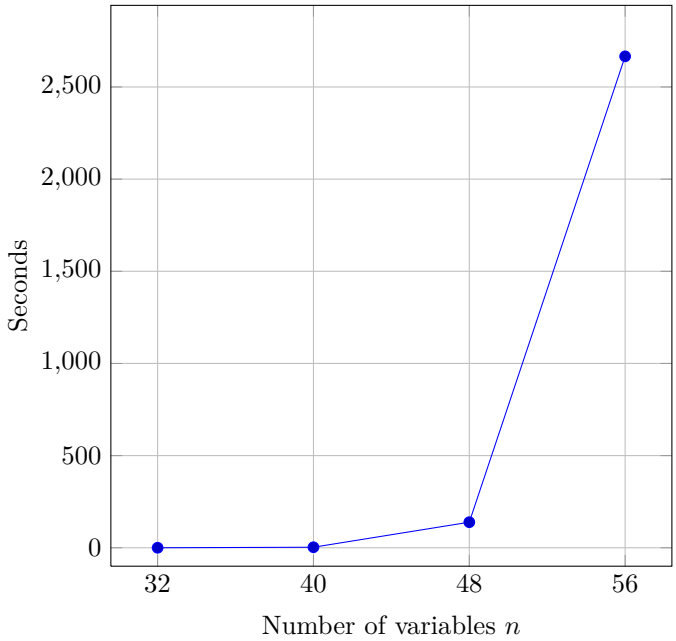


Figure 5.1: Average run times for 100 MQQ-SIG instances of n variables, with $\frac{n}{2}$ equations removed from the public key.

Table 5.1: The average time to compute the Gröbner basis for 100 different MQQ-SIG instances in n variables, and with $\frac{n}{2}$ equations removed. All numbers in seconds.

n	$\bar{t}$	σ
32	0.13	0.06
40	2.88	0.78
48	138.75	45.66
56	2666.15	739.92

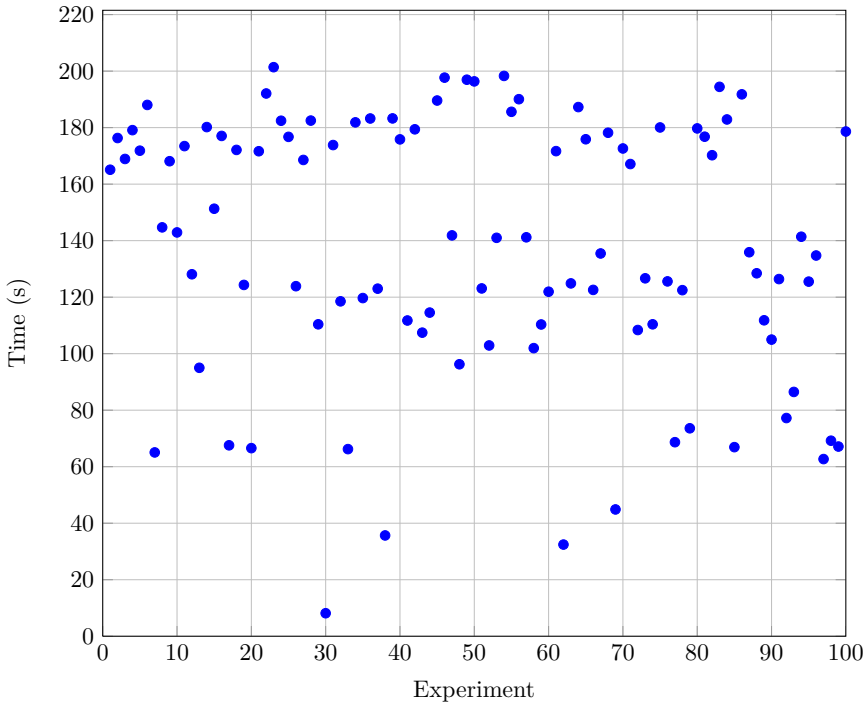


Figure 5.2: The time to compute the Gröbner basis for 100 different public keys of 48 variables.

In Table 5.2, we present the average degree of regularity that appeared during the runs of the experiments. We have also included the results of running the system with fewer than $\frac{n}{2}$ equations removed. There are some missing values in the table for the case $n = 56$, indicated by the dots. For these specific scenarios, many of the instances either required more memory than our computer had, or did not finish within the time frame set for our experiment.

What these results show, is that the degree of regularity increase, both when we add more variables to the system, and when we remove more equations from the public key. MQQ-SIG appears to follow the theorized behavior of a random set of multivariate equations. Recall from Section 5.1, that the original MQQ cipher was broken because the degree of regularity did not increase with more added variables. We see that MQQ-SIG does not share this property.

We also mention that Gligoroski et al. [GØJ+11] also conducted a similar type of experiment on the degree of regularities of MQQ-SIG. Their findings are essentially the same as ours.

5.5 The matrix $U(\mathbf{x})$

The importance of analyzing the matrix $U(\mathbf{x})$ is apparent since it is the sole factor in determining which linear isotopy class the quasigroup will belong to. To make this more precise, recall from Definition 3.17 that a linear isotopy between quasigroups is a triple of bijective linear maps, and from Theorem 3.18 that a linear isotopy represents an equivalence relation on the set of quasigroups. Hence, in MQQ-SIG, $U(\mathbf{x})$ defines a specific isotopy class, while A_1, A_2 and B functions as the isotopy triples. They send the quasigroup created by $U(\mathbf{x})$ to different quasigroups within the same equivalence class. In MQQ-SIG these classes are linear, that is, defined by linear permutations, but we remark that isotopies do not have to be linear in general.

Since $U(\mathbf{x})$ is the single source for different isotopy classes, it is interesting investigate the characteristics of the classes it might create. Recall that $U(\mathbf{x})$ is an upper triangular matrix containing linear expressions in $\mathbf{x}$. If a row does not contain any terms in $\mathbf{x}$ (more specifically: all elements in the row except for the diagonal element is zero), we call it a *linear row*.

Table 5.2: The average degree of regularity for 100 MQQ-SIG instances in n variables and r equations removed. In parentheses: the expected degree of regularity for a random system of size $n - r$.

r/n	16	24	32	40	48	56
8	**3 (3)**	3.58 (4)	3.94 (5)	4.10 (6)	4.06 (7)	4.02 (8)
9		3.65 (4)	3.97 (5)	4.14 (6)	4.17 (7)	4.32 (8)
10		3.59 (4)	3.98 (5)	4.09 (6)	4.19 (7)	4.37 (8)
11		3.61 (4)	4.00 (5)	4.28 (6)	4.21 (7)	4.58 (8)
12		**3.69 (4)**	4.00 (5)	4.68 (6)	4.72 (7)	4.78 (8)
13			4.00 (5)	4.84 (6)	4.96 (7)	4.95 (8)
14			4.00 (5)	4.92 (6)	5.00 (7)	5.02 (8)
15			4.00 (5)	4.96 (6)	5.00 (7)	5.00 (8)
16			**4.00 (5)**	4.97 (6)	5.00 (6)	5.25 (8)
17				4.98 (5)	5.03 (6)	5.20 (7)
18				4.99 (5)	5.08 (6)	- (7)
19				5.00 (5)	5.19 (6)	- (7)
20				**5.00 (5)**	5.32 (6)	- (7)
21					5.47 (6)	- (7)
22					5.62 (6)	- (7)
23					5.61 (6)	- (7)
24					**5.45 (6)**	- (6)
25						- (6)
26						- (6)
27						5.99 (6)
28						**5.99 (6)**

Observation 5.10. Every generated $U(\mathbf{x})$ contains at least one linear row, namely the last one. This is easy to see, since $U(\mathbf{x})$ is an upper triangular matrix, where the last row is of the form: $(0\,0\,\cdots\,0\,1)$. Hence, the last coordinate in the product $U(\mathbf{x}) \cdot \mathbf{y}$ will contain only linear expressions in $\mathbf{y}$, justifying the term *linear row*. Without the isotopy (A_1, A_2, B), every MQQ in MQQ-SIG will have at least one of its polynomials containing no quadratic terms.

It is undesirable with an MQQ that contains only linear terms in some of its coordinates, since it reduces the resulting system complexity. Unfortunately, simply specifying that that the quasigroup should contain no linear coordinates is not enough to guarantee a strong system. Gligoroski et al. [GØJ+11] points out that the quadratic polynomials of an MQQ of type $\mathrm{Quad}_{d-k}\mathrm{Lin}_k$, might cancel each other when combined linearly, yielding a system of less then $d - k$ quadratic polynomials.

In other words, the isotopy (A_1, A_2, B) whose role we can consider to be that of distributing the quadratic terms among the MQQ vector, might actually weaken the system.

Later Chen et al. [CKG10] ascertained that this was in fact the case; even more so, it also happened more frequently than previously expected. They concluded that the $\mathrm{Quad}_{d-k}\mathrm{Lin}_k$ classification of MQQ's is a poor way of assessing its true complexity. Accordingly, they suggested an alternative way of classifying the MQQ's; introducing the notion of a strict MQQ type.

Definition 5.11. A quasigroup $(Q, *)$ of order 2^d is called an *MQQ of strict type*, denoted $\mathrm{Quad}^{\mathrm{s}}_{d-k}\mathrm{Lin}^{\mathrm{s}}_k$, if there are at least $d - k$ quadratic polynomials whose linear combinations do not result in a linear form, where $0 \leq k < d$.

The definition of $U(\mathbf{x})$ in MQQ-SIG is based on a procedure for creating quasigroups of strict types, taken from [CKG10]. The number of linear rows in $U(\mathbf{x})$ correspond to the number k in the definition of strict MQQ's. In particular, the quasigroups in MQQ-SIG are of the type $\mathrm{Quad}^{\mathrm{s}}_7\mathrm{Lin}^{\mathrm{s}}_1$ for one linear row, $\mathrm{Quad}^{\mathrm{s}}_6\mathrm{Lin}^{\mathrm{s}}_2$ for two linear rows, and so on. The specification does not allow for the creation of MQQ's of type $\mathrm{Quad}_8\mathrm{Lin}_0$. How to construct quasigroups of this type is an open problem.

To evaluate the effect of having linear rows in $U(\mathbf{x})$, we created several quasigroups where $U(\mathbf{x})$ had between one to six linear rows, and measured their performance on a system with $n = 48$ variables. The exact procedure was as follows:

1. Create *one* set of random values for $A_1, A_2, B, \mathbf{c}, S$ and $\mathbf{v}$. For simplicity we chose to generate S and $\mathbf{v}$ at random, not using the special construction described in Section 4.3.

2. Generate 40 $U(\mathbf{x})$ matrices with one linear row, 40 matrices having two linear rows, and so on[§].

3. Run Algorithm 5.1 on the 40 instances of each number of linear rows. Note that the same set of values, created in Step 1, was used for all experiments. That is, only the $U(\mathbf{x})$ matrix changed during the procedure, while all other values was kept constant.

[§]More explicitly: we simply created random $U(\mathbf{x})$ matrices repeatedly until we found one with one linear row. Then we manually set the last k rows to be linear rows, with $1 \leq k \leq 6$.

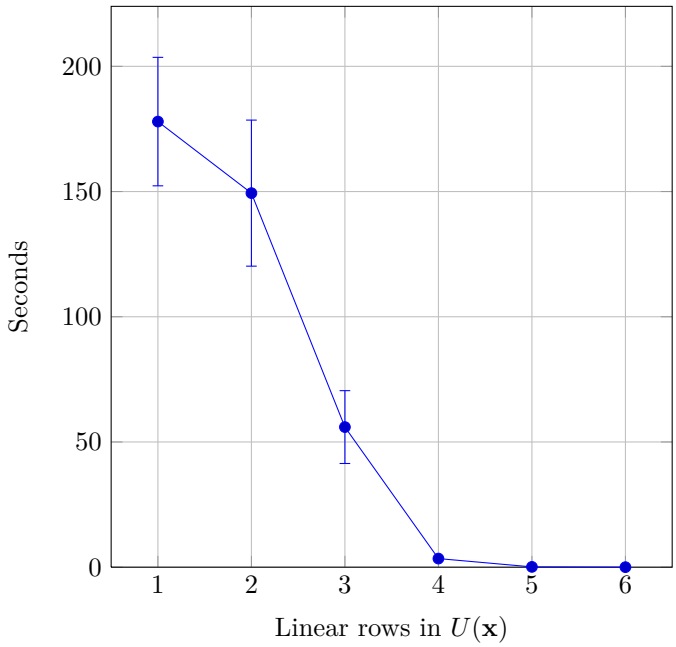


Figure 5.3: The effect of having linear rows in the $U(\mathbf{x})$ matrix. The graph shows the average run time of 40 different instances of 48 variables, all having the same number of linear rows in $U(\mathbf{x})$.

The results are shown in Figure 5.3 and clearly demonstrates that more linear rows in $U(\mathbf{x})$ affect the behavior of the system in a negative way. It is rather conclusive that the $U(\mathbf{x})$'s with several linear rows should be avoided, while those with only one linear row should be preferred. It should be noted however, that the probability of a randomly created $U(\mathbf{x})$ to have more than three linear rows is relatively low. Two or three on the other hand, occur fairly often. Still, the vast majority contains only one linear row, so adding this criterion to the creation of the $U(\mathbf{x})$ matrices does not add many extra cycles to the key generation algorithm.

Recommendation 1. The isotopy classes used in MQQ-SIG should be those that are determined by $U(\mathbf{x})$ matrices having only one linear row.

5.6 The matrices A_1 and A_2

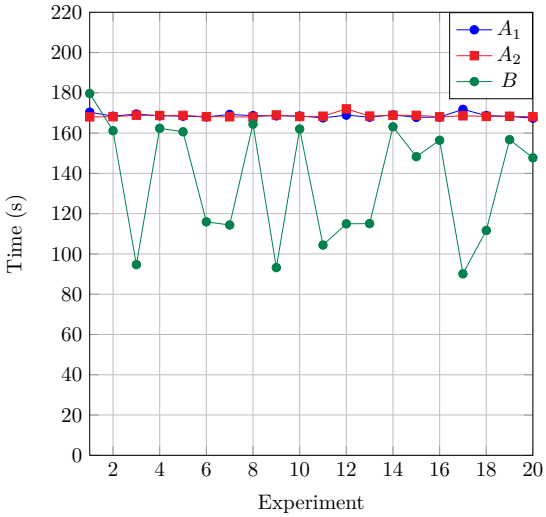
Fix an isotopy class by choosing some $U(\mathbf{x})$. Now it is interesting to see how a linear isotopy affects the properties of the quasigroups within this equivalence class. As shown in Proposition 3.19 the degrees of quasigroups are invariant under linear isotopies. If the security against Gröbner basis attacks also remained the same it would alleviate the need for the matrices A_1, A_2 and B. On the other hand, if the strength of the system varies with different isotopies, we should investigate their features to understand why they might lead to weak keys. To this end, we examine the affects of the matrices A_1 and A_2 in this chapter and look at the matrix B in the next.

To gauge the contribution of either A_1, A_2 or B, we picked out three instances of 48 variables from the original experiments (see Section 5.4). We chose the instances so as to include one with very good performance, one with bad performance and one being close to the average of all the 100 instances. For ease of reference, we will refer to them as the "Good", the "Bad" and the "Average" instance, respectively. For all three instances, we derived 20 new experiments for each matrix A_1, A_2 and B. That is, for each instance we generated 20 new values for A_1 and used them to create new public keys where all values, except A_1, remained the same as in the original instance. Similarly, for A_2 and B. We then ran Algorithm 5.1 on all these modified instances.

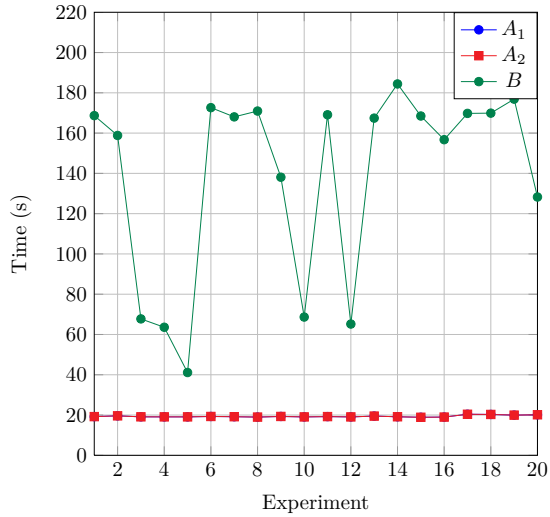
The results are shown in Figure 5.4, where 5.4a represents the "Good" instance on 20 different values for A_1, A_2 and B, Figure 5.4b the "Bad" instance, and Figure 5.4c the "Average" instance. As apparent from these graphs, changing either A_1 or A_2 do not to alter the behavior of the given isotopy class. For all three cases the performance is highly consistent across the 20 different values of A_1 and A_2. More importantly, a different value of either A_1 or A_2 will not radically change the behavior of the quasigroup. If given a weak quasigroup by $U(\mathbf{x})$ and B, changing A_1 or A_2 will not help to make it stronger. This motivates the following conjecture.

Conjecture 5.12. *Fix $U(\mathbf{x})$, B and $\mathbf{c}$ in Equation 4.1. Then the equivalence class of quasigroups generated by the isotopy (A_1, A_2, B), for randomly generated A_1 and A_2, is invariant with respect to their security against Gröbner basis attacks.*

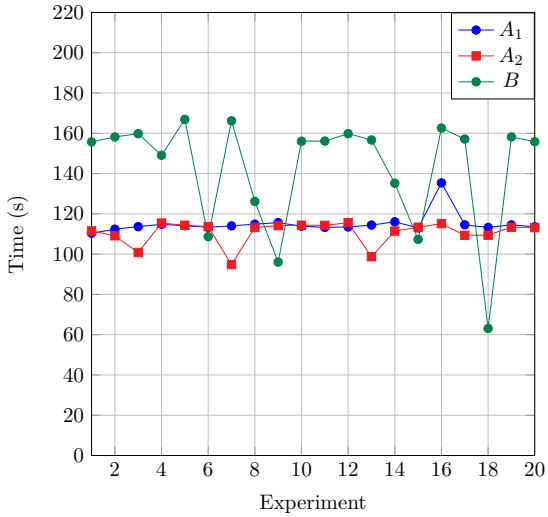
The matrix B, on the other hand, is seen to heavily affect the run times of the



(a) The "Good" instance



(b) The "Bad" instance



(c) The "Average" instance

Figure 5.4: The run times of three instances of 48 variables from the original experiments in Section 5.4, using 20 different values for the matrices A_1, A_2 and B.

Gröbner basis computations. Additionally, B has the ability to drastically change the behavior of a quasigroup. In Figure 5.4 we see that for some values of B, the "Good" instance performs very bad, and on others again, the "Bad" instance is much better. This indicates that there are some decisive attributes of B which influence the performance of the system. Hence, subdividing the isotopy class based on different B's, seems to be most relevant when classifying the quasigroups (up to linear isotopies). We will study B in more detail in the next section.

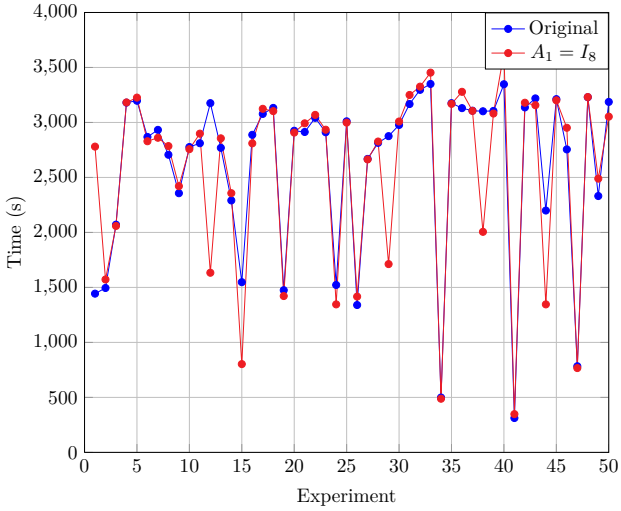
To further validate the irrelevance of A_1 and A_2 with respect to resistance against Gröbner basis attacks, we ran several experiments where we simply left out A_1 or A_2. For all the instances of 40, 48 and 56 variables generated in Section 5.4, we reran Algorithm 5.1, but with the matrices A_1, A_2 set to the identity matrix, either separately or together. That is, for each instance of a given number of variables n, we ran three experiments: one with $A_1 = I_8$, one with $A_2 = I_8$, and one with $A_1 = A_2 = I_8$. All the other parameters remained the same as in the original experiment.

The average times are shown in Table 5.3, but for a clearer exposition we also plot the 50 first instances of the case $n = 56$, with $A_1 = I_8$ and $A_2 = I_8$, in Figure 5.5a and 5.5b, respectively. We do not show all the instances since the 50 first are sufficient to clearly illustrate the result, and because the remaining instances exhibited identical behavior (also for $n = 40$ and 48).

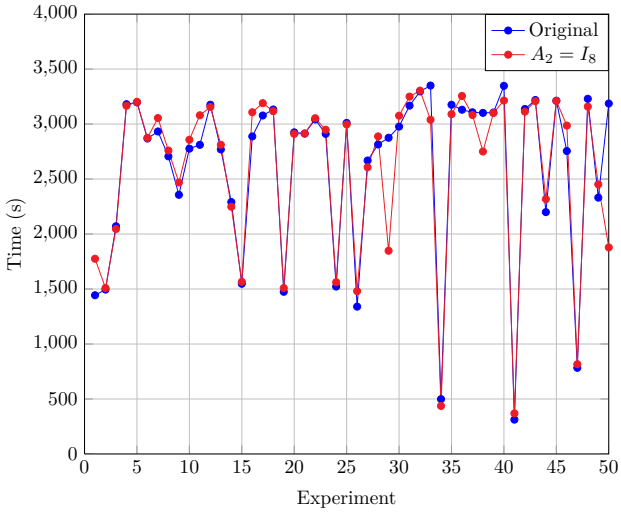
Table 5.3: Average run times for computing the Gröbner basis of 100 MQQ-SIG instances of n variables, with A_1 and A_2 set to the identity matrix. All times are in seconds.

		Matrix removed		
n	Original	A_1	A_2	A_1 and A_2
40	2.88	2.70	2.75	2.60
48	138.75	130.03	129.55	119.51
56	2666.15	2609.95	2643.38	2553.40

As these two graphs distinctly show, removing A_1 or A_2 from the MQQ construction yields a system with virtually identical behavior. Except for a few outliers, which are probably due to some rare event of interplay with the other system parameters, the plots are almost indistinguishable. This is also further supported



(a) $A_1 = I_8$



(b) $A_2 = I_8$

Figure 5.5: The run times of the first 50 original instances of 56 variables where either A_1 or A_2 is set to the identity matrix

by the average times given in Table 5.3. Since A_1 and A_2 are unnecessary for the security of an MQQ, we make the following recommendation:

Recommendation 2. The matrices A_1 and A_2 can be excluded from the definition of a quasigroup in MQQ-SIG (Equation 4.3), without affecting the security of the system against Gröbner basis attacks.

5.7 The matrix B

We saw in the previous section that the matrix B is very influential for the behavior of a quasigroup. The significance of B can be seen from the expression

$$B \cdot (U(\mathbf{x}) \cdot A_2 \cdot \mathbf{y}),$$

which is taken from the equation defining the MQQ's in MQQ-SIG (cf. Equation 4.3). Because $U(\mathbf{x}) \cdot A_2 \cdot \mathbf{y}$ is a vector of quadratic terms, B will be a decisive factor in how those terms will be combined linearly in the resulting MQQ. This relates it to the ranks of the $\mathbf{B}_{f_i}$ matrices, which we mentioned in Section 4.2.1 might actually be important for Gröbner basis computations.

Recall that there were two additional criteria on the quasigroups to be used in MQQ-SIG. Namely, all the eight matrices $\mathbf{B}_{f_1}, \mathbf{B}_{f_2}, \ldots, \mathbf{B}_{f_8}$, must have rank at least 12, and at least one matrix must have rank 14 (see Equation 4.5a and 4.5b, respectively), where the $\mathbf{B}_{f_i}$ corresponds to the quadratic terms of polynomial f_i in the MQQ. Since B and $U(\mathbf{x})$ are the parameters responsible for the quadratic terms appearing in MQQ-SIG, the ranks of the $\mathbf{B}_{f_i}$'s will be totally dependent on these two matrices. For a Gröbner basis algorithm, the number of quadratic terms in the polynomials can be an important factor for the effectiveness of the algorithm. Thus, the criteria in Equation 4.5a and 4.5b, which were initially intended to defeat MinRank attacks, might actually be important against Gröbner basis attacks as well.

Since MinRank attacks were outside the scope of this thesis, we did not enforce the requirements when creating the quasigroups in the original experiments. Based on the above, this could have had a detrimental effect on the strength of the generated keys.

To assess the prevalence of instances fulfilling the requirements naturally, we tested each of the original instances to see if they satisfied the criteria. Of all the experiments we had created (of all number of variables), only four fulfilled the requirements, that is, only one in one hundred. Even more, we found that many instances had $\mathbf{B}_{f_i} = 0$ for some of their polynomials f_i. What this means, is that such quasigroups would have some of their polynomials, f_i, without quadratic terms. The MQQ's would be of type $\mathrm{Quad}_7\mathrm{Lin}_1$ (one linear function) or $\mathrm{Quad}_6\mathrm{Lin}_2$ (two linear functions). Public keys based on such instances will most likely contain fewer quadratic terms after the mixing stage with $\mathcal{S}$, than those of type $\mathrm{Quad}_8\mathrm{Lin}_0$. Consequently, the Gröbner basis algorithm will probably be able to find relations among the polynomials quicker, and thus eliminate terms faster. Conversely, if the quasigroups were to satisfy the requirements in Equation 4.5a and 4.5b, they would most likely exhibit some minimal amount of complexity.

Therefore, we chose test the real influence the ranks of the $\mathbf{B}_{f_i}$ matrices could have on Gröbner basis computations. We tested three cases of MQQ's: (1) MQQ's where at least one of the $\mathbf{B}_{f_i}$ matrices had rank zero; (2) MQQ's where all satisfied the specified criteria; (3) MQQ's where no $\mathbf{B}_{f_i}$ matrix had rank zero, but did not also have to satisfy the specified criteria.

Case 1 was to confirm that MQQ's with zero-ranks would most probably perform badly. Case 2 was to compare the results of satisfying the criteria with our original experiments, where no checks had been performed. Case 3 was included to account for the possibility that the systems satisfying the criteria, might see better results the original instances simply because they did not include any MQQ's with linear polynomials.

The quasigroups were derived from the original set of systems in Section 5.4, by randomly creating new B matrices until the criteria were met. All other parameters of the systems remained the same. We ran all 100 (adjusted) instances for 40, 48 and 56 variables and calculated their average results. The times are summarized in Table 5.4.

The table seems to confirm our hypothesis that quasigroups of type $\mathrm{Quad}_7^{\mathrm{s}}\mathrm{Quad}_1^{\mathrm{s}}$ and $\mathrm{Quad}_6^{\mathrm{s}}\mathrm{Quad}_2^{\mathrm{s}}$, should perform noticeably worse than other instances. For $n = 48$ or $n = 56$, their results are approximately half that of the original systems, or around two standard deviations worse. On the other hand, the MQQ's that satisfy

Table 5.4: Average run times for 100 MQQ-SIG instances of n variables, conditioned on the matrices $\mathbf{B}_{f_i}$. All times in seconds.

	No conditions (original system)		At least one rank zero		No rank zero		All satisfy Equation 4.5	
n	$\bar{t}$	σ	$\bar{t}$	σ	$\bar{t}$	σ	$\bar{t}$	σ
40	2.88	0.75	1.90	0.93	2.86	0.72	2.82	0.81
48	138.75	45.66	74.94	54.93	142.39	42.62	138.38	46.09
56	2666.15	709.56	1434.31	894.83	2593.06	711.56	2721.13	637.70

Equation 4.5a and 4.5b, do not appear to perform significantly better than the "no-zero-rank"-experiments. This suggests that the main advantage of criteria in Equation 4.5 when it comes to resistance against Gröbner basis attacks, is to filter out the quasigroups with linear polynomials.

From an efficiency point of view, there is a clear advantage in only requiring that the quasigroups satisfy the "no-zero-rank" criterion. Since this is a much weaker requirement, probably a lot more quasigroups will satisfy it, hence generation times will be faster. To quantify this, we also recorded the number of different B matrices we had to try before finding a suitable MQQ, our findings are shown in Table 5.5.

Table 5.5: The average number of B matrices to test before satisfying the two rightmost criteria in Table 5.4.

n	No rank zero	All satisfy Equation 4.5
40	1.12	400.00
48	1.06	322.02
56	1.25	434.64

As apparent from this table, the sheer number of B's that have to be tested before satisfying Equation 4.5, means that this can potentially be a bottle-neck in the key generation algorithm. As mentioned in Section 4.4.2, the designers of MQQ-SIG already acknowledged that the key generation would most likely be the most demanding part of the system. By removing these unnecessary criteria and only filter out the "zero-rank" instances instead, we can hope to see speed improvements in the key generation by a large factor. Hence, we end with the

following recommendation.

Recommendation 3. The criteria in Equation 4.5 can be replaced with the following:

$$\forall i \in \{1, \ldots, d\},\ \mathrm{Rank}(\mathbf{B}_{f_i}) > 0.$$

In other words, it is only required that all elements in the MQQ-vector contain quadratic terms.

5.8 The matrix S

The last parameter of MQQ-SIG we consider is the matrix S. Recall that S defines the linear and affine transformations $\mathcal{S}$ and $\mathcal{S}'$ respectively. Their purpose is to hide the internal structure of the central map $\mathcal{P}'$. In most $\mathcal{MQ}$-schemes these linear/affine transformations are simply created by choosing two random matrices over the field relevant to the scheme. Since a random matrix will not exhibit any internal exploitable structure (given that it has been created by a sufficiently strong pseudo-random generator), it can be considered the optimal choice for a linear transformation in terms of security. However, this means that the entire matrix has to be stored, which adds to the already large key sizes found in $\mathcal{MQ}$-schemes.

Recall from Section 4.3 that the designers of MQQ-SIG tried to mitigate this problem by creating S in very special manner[¶]. S is created as the combination of two circulant matrices, which makes it is possible to avoid storing the full matrix, but simply the two permutations σ_0^0 and σ_0^1.

The reason for using two circulant matrices instead of just one, was to avoid the regular structure exhibited in circulant matrices. Even so, the current construction is still highly structured. Ideally we would hope that S resembles a truly random matrix, but there is no guarantee that none of its internal structure might leak through in a Gröbner basis attack. Thus, we decided to evaluate the behavior of S compared to a truly random matrix.

Our experiment procedure was similar to the one in previous sections: first we generated a random matrix S for each of the 100 original instances with 40, 48 and

[¶]Actually, it is S^{-1} that is created in this way, and S is derived from it as its inverse, i.e. $S = (S^{-1})^{-1}$. This distinction is not important here, so we will simply refer to the matrix S.

56 variables. Then, we used these matrices to derive new instances from the original ones; keeping all parameters the same except for the matrix S. The average results of running Algorithm 5.1 on these modified instances are shown in Table 5.6. We also plot the graph for the 50 first instances for 48 and 56 variables[††], to better illustrate the behavior in Figure 5.6.

Table 5.6: Average run times for 100 MQQ-SIG instances of n variables, with random S

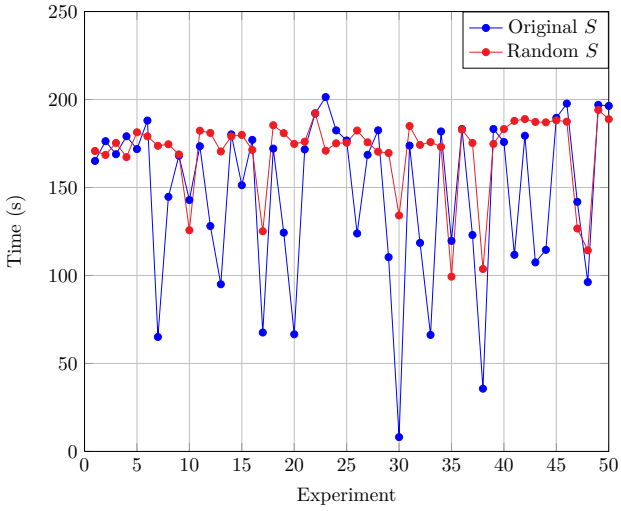
	Original		Random S	
n	$\bar{t}$	σ	$\bar{t}$	σ
40	2.88	0.75	3.22	0.45
48	138.75	45.66	167.44	24.78
56	2666.15	709.56	2996.41	227.64

We see that the modified instances, using a random S, performed better then the original experiments. While comparable, the difference of their average results is noticeable. What is more interesting to note however, is how much more consistent the behavior of the systems with random S are. Figure 5.6 demonstrates that the original instances have many instances performing very badly, and that there in general is a large variation in their results. With random matrices, the run times seem to converge towards a common value, having only a few deviating results. And even then, those few instances do not lead to extremely bad keys, but are still relatively close to the average.

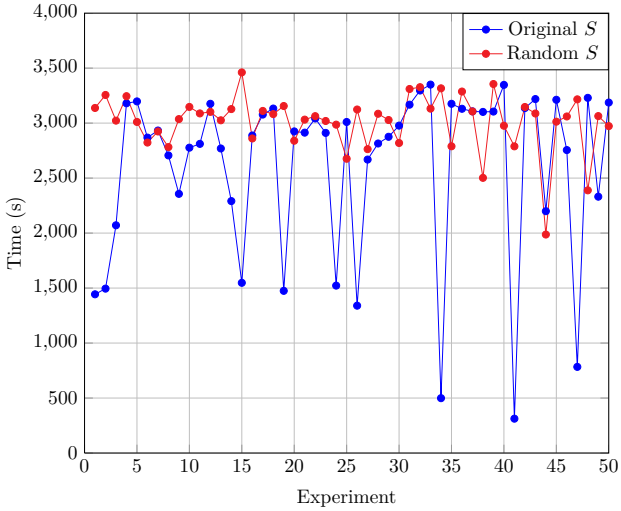
These results indicate that the specially constructed S is not able to hide the internal algebraic structure of the system quite as well as a randomly created S. This supports our initial assessment of random matrices being the optimal choice for the linear transformations in terms of hiding the central map $\mathcal{P}'$. In some sense we can regard the random construction as the benchmark which other designs should be measured against.

While the construction used in MQQ-SIG is comparable to the performance of random matrices on average, it still suffers from the occurrence of very bad keys. If this design is still to be used, it is essential to be able to identify the instances that

[††]We chose not to show the graph of the case $n = 40$, since the run times of these instances are so low. Nevertheless, they displayed the same characteristic behavior as seen in Figure 5.6.



(a) 48 variables



(b) 56 variables

Figure 5.6: The run times of the first 50 original instances of 48 and 56 variables with a random S matrix

might lead to weak keys, so they can be rejected during key generation. To this end, we tried to investigate some of the parameters that constitute the matrix S in MQQ-SIG, in order to determine their influence on the systems behavior.

Recall from Proposition 4.4 that the matrix S^{-1} (and hence S) can be uniquely determined solely by the parameters σ_0^0 and σ_0^1, which are permutations on n symbols. All permutations can be decomposed into a product of disjoint cycles of length $\leq n$. We chose to focus on this decomposition and looked specifically on the cycle lengths. From the original experiments on 48 variables in Section 5.4, we selected the five best and the five worst performing instances, and created two experiments:

1. For each instance, create a new S where the permutations σ_0^0 and σ_0^1 are chosen so that one of the cycles in their disjoint cycle decomposition is of length ≥ 45. Create 20 such matrices for each instance and measure their performance.

2. Similar to the experiment above, but now chose σ_0^0 and σ_0^1 so that none of the cycles in their disjoint cycle decomposition have length greater than six.

In essence, we had one experiment where the S matrices were created from permutations of very long cycle lengths, and one where they were created from very short cycle lengths. This is of course a very crude test procedure, but we chose it due to its simplicity, and because our main focus has been on the quasigroups of MQQ-SIG. These tests were just a simple means of discovering whether there existed a relationship between the cycle lengths and the performance of the linear transformations. More elaborate test could have been conducted, but we considered it outside the scope of this thesis. Unfortunately, the results did not give any conclusive evidence in any direction. The experiments where σ_0^0 and σ_0^1 contained long cycles performed neither better nor worse than the original instances in any consistent manner. Likewise for the case of short cycles. In short, our experiments were not sufficient to establish any clear connection between the lengths of the permutation cycles and the performance of the resulting system. Analyzing other properties of the S matrix in MQQ-SIG could be the topic of further research.

To conclude, we have found that using the special construction of S in MQQ-SIG, performs worse than using a random matrix. While the average results where

comparable, weak keys were generate relatively frequently when using this design, something that did not happen when using random matrices. We have not been able to determine exactly which properties of the S matrix in MQQ-SIG that differentiates it from a random matrix, but there is a definite difference.

That leaves the question whether this construction should be used or not. The main reason for wanting to use it, is because of the significant reduction in private key sizes that is possible. Incidentally, we only compared the use of random matrices with the original instances, and as we have seen in the previous sections, they contained many bad MQQ's. Hence, it might be conceivable that if these quasigroups were filtered out, we might see better results from the special design. Additionally, the expression defining the matrix S (cf. Equation 4.8) is specifically designed to be very efficient in both hardware and software [GØJ+11]. On the other hand, if the division operators of the MQQ are implemented using pre-computed lookup tables, then the additional cost of storing a full random matrix is negligible.

In the end, the strong and consistent performance of using a random S, and the uncertainty of why the special construction behaves the way it do, we are led to the following recommendation.

Recommendation 4. The matrix S, and the vector $\mathbf{v}$, should be created uniformly at random.

CHAPTER 6

Enhancing the MQQ-SIG Key Generation Algorithm

In the previous chapter we offered four concrete recommendations on how the various parameters of MQQ-SIG should be created in order to avoid weak keys from being generated. However, they are based on measurements were we had adjusted a single system parameter individually. In the following short chapter, we will put all these recommendations together into a coherent whole and propose an enhanced key generation algorithm for MQQ-SIG. Furthermore, we provide numerical results on the keys created by this new algorithm and compares them to the experiments we did on original MQQ-SIG specification. The results conclusively show that the modified key generation algorithm performs much better than the original.

6.1 The new key generation algorithm

First we summarize the results from the previous chapter and the corresponding recommendations we derived from them.

1. More linear rows in $U(\mathbf{x})$ lead to worse performance. Consequently, when creating $U(\mathbf{x})$ it should be checked that it only contains one linear row (Recommendation 1).

2. Including A_1 and A_2 does not yield a stronger system. Hence, we will simply drop them in the new algorithm (Recommendation 2).

3. MQQ's of type $\text{Quad}_8\text{Lin}_0$ perform better than quasigroups where one or more of the coordinates of the MQQ-vector contain only linear expressions. Thus we should ensure that only $\text{Quad}_8\text{Lin}_0$ quasigroups are used (Recommendation 3). We achieve this by generating new B matrices until the requirement is met.

4. Using a random S matrix as the basis for the transformations $\mathcal{S}$ and $\mathcal{S}'$ gives better and more consistent behavior of the system. Therefore, we will not use the original design with circulant matrices, but simply use random matrices (Recommendation 4).

The modified key generation algorithm, taking these recommendations into account, is given in Algorithm 6.1. Note that the way the parameters are used to construct the public and private key has not changed; additionally, the verification and signing procedure are just the same as in the original specification.

6.2 Performance

6.2.1 Procedure

Using the 100 instances from Section 5.4 as a basis (which were created according to the original MQQ-SIG specification), we created modified instances as per Algorithm 6.1, with 40, 48 and 56 variables. More specifically, given one of the original instances, we first set A_1 and A_2 equal to the identity matrix. Then, we checked if the original $U(\mathbf{x})$ had more than one linear row; if not, we simply used it as is; else we generated a new one until a suitable matrix was found. With the $U(\mathbf{x})$ matrix selected, we used the original B matrix to create an MQQ according to Equation 6.1, and computed the ranks of its $\mathbf{B}_{f_i}$ matrices (cf. Equation 4.6). If none of the matrices had rank 0 (Recommendation 3), we used the resulting MQQ, else we created a new B matrix until the criterion was met. Finally, we created a

Algorithm 6.1 Enhanced MQQ-SIG key generation

Input: The number of variables n, where $n \in \{160, 192, 224, 256\}$.
Output: A public key $\mathcal{P}$ given by $\frac{n}{2}$ multivariate quadratic polynomials $P_i(x_1, \ldots, x_n)$, $i = 1 + \frac{n}{2}, \ldots, n$. A private key consisting of a MQQ of order 2^8, a random invertible $n \times n$ matrix S over $\mathbb{F}_2$, and a a random vector $\mathbf{v}$ in $\mathbb{F}_2^n$.

The central map $\mathcal{P}'$
The MQQ is defined by the following expression:

$$\mathbf{x} * \mathbf{y} = B \cdot U(\mathbf{x}) \cdot \mathbf{y} + B \cdot \mathbf{x} + \mathbf{c}, \tag{6.1}$$

where all the parameters have the same meaning as in Equation 4.3. Additionally, $U(\mathbf{x})$ and B are subject to the following restrictions:

- $U(\mathbf{x})$ must have no linear rows other than the last;
- B must be chosen so that the resulting MQQ is of type $\text{Quad}_8\text{Lin}_0$, or in other words, $\text{Rank}(\mathbf{B}_{f_i}) > 0$ for all $1 \leq i \leq 8$, where the $\mathbf{B}_{f_i}$ matrices are defined as in Equation 4.6.

Finding $U(\mathbf{x})$ and B that satisfy these constraints, can be done efficiently by simple trial-and-error until the requirements are met. The central map $\mathcal{P}'$ is constructed in exactly the same way as in the original algorithm (cf. Algorithm 4.2).

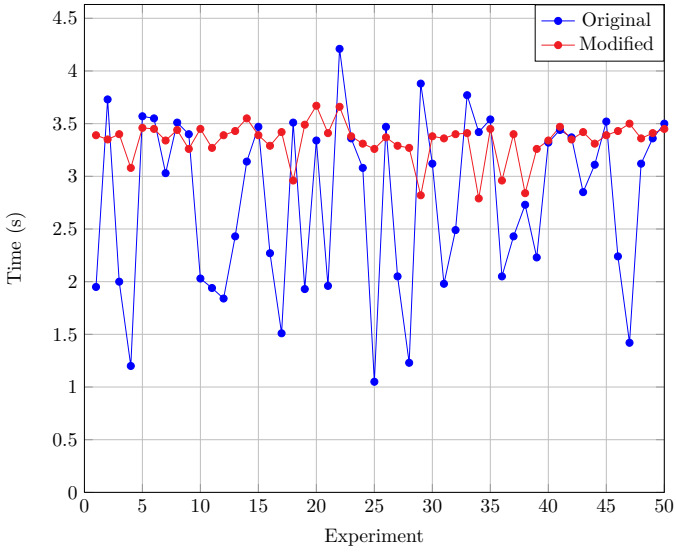
The linear transformations $\mathcal{S}$ and $\mathcal{S}'$
The matrix S is constructed as a random invertible $n \times n$ matrix over $\mathbb{F}_2$, and $\mathbf{v}$ is constructed as random element in $\mathbb{F}_2^n$. $\mathcal{S}$ and $\mathcal{S}'$ are then defined just as in the original algorithm (cf. Equation 4.9 and Equation 4.10, respectively).

---

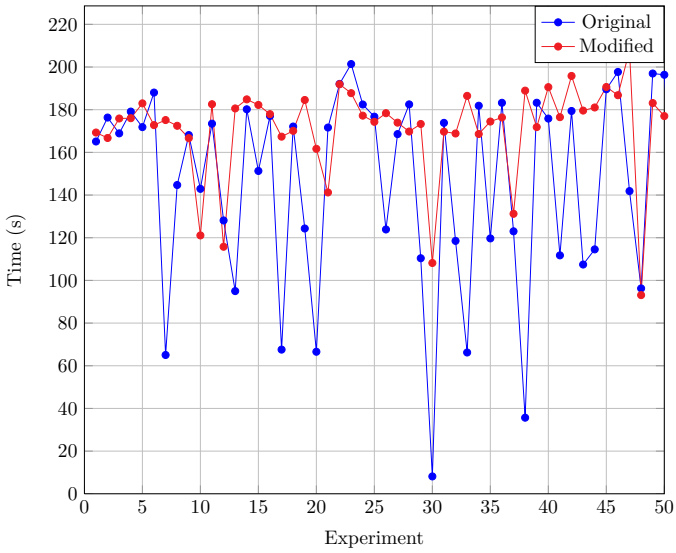
random matrix S and a random vector $\mathbf{v}$, for the linear and affine transformations $\mathcal{S}$ and $\mathcal{S}'$.

6.2.2 Results

In Figure 6.1 we illustrate the run times of the 50 first instances from the original experiments in Section 5.4 ($n = 40, 48$ and 56), modified according to the procedure given above. We only plot the first 50 instances, since the remaining instances showed exactly the same behavior. However, Table 6.1 summarizes the average run times of all 100 experiments.

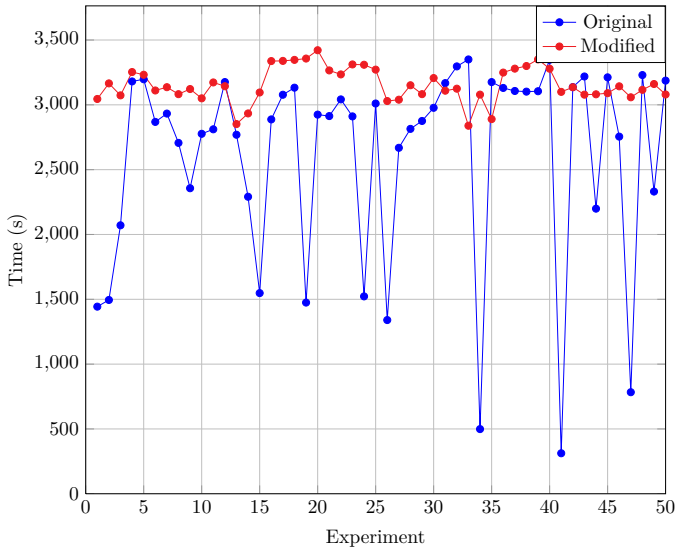


(a) 40 variables



(b) 48 variables

Figure 6.1: Run times of 50 instances based on the original experiments in Section 5.4, adjusted to conform to Algorithm 6.1.



(c) 56 variables

Figure 6.1: (continued)

Table 6.1: Average run times for 100 MQQ-SIG instances of n variables, using the modified key generation algorithm in Algorithm 6.1

	Original		Modified	
n	$\bar{t}$	σ	$\bar{t}$	σ
40	2.88	0.75	3.33	0.25
48	127.49	45.66	161.15	18.49
56	2666.15	709.56	3127.55	143.93

These results clearly show that the MQQ-SIG instances that were modified in order to comply with Algorithm 6.1, perform significantly better than the original instances. As in the experiments where we focused individually on the S matrix (Section 5.8), we see that the modified approach performs much more consistently than the original key generation. However, this effect is even more profound now, when we include the changes to the MQQ parameters, than when simply changed to a random S matrix, but kept the MQQ generation the same. We both have smaller fluctuations between the generated instances, and higher run times on average. This is a testament to the importance of creating strong MQQ's as well. Furthermore, note that with an increasing number of variables, the "weakest" keys generated are relatively much closer to the average, than with fewer variables (contrast, for example, Figure 6.1b with Figure 6.1c).

If this alleged behavior continues until the level of a real MQQ-SIG system, that is, with $n = 160, 192, 224$ or 256, it would mean that very weak keys are unlikely to be generated by Algorithm 6.1. In the end, these results are strong evidence in support of modifying the MQQ-SIG key generation algorithm to the one in this chapter.

CHAPTER 7

Conclusion

MQQ-SIG is a promising new signature scheme that vastly outperforms today's public-key signature systems in terms of signing and verification speed, albeit at the cost of very large public keys. Also, when compared to other $\mathcal{MQ}$-schemes, its signing speed is significantly better. This is due to the efficiency of the central map $\mathcal{P}'$, which is based on multivariate quadratic quasigroups (MQQ's). However, not much cryptanalysis has been performed on this special design. In particular, it is not known whether the strong algebraic properties of quasigroups affect the structure of the resulting public key. The contribution of this thesis has been a first step in that direction.

Our main result is a classification of the MQQ's that leads to weak keys in MQQ-SIG, based on empirical evidence from running numerous experiments on the system. We have identified that the most important parameters of an MQQ, when it comes to security against Gröbner basis attacks, are the matrices $U(\mathbf{x})$ and B. The matrix $U(\mathbf{x})$ is responsible for which linear isotopy class the MQQ will belong to, and therefore is a key factor in determining the base properties of the quasigroup. For maximal complexity, we have found that $U(\mathbf{x})$ should not have more than one linear row.

Then, given that a linear isotopy class has been selected (governed by some choice of $U(\mathbf{x})$), the three matrices A_1, A_2 and B define the potential quasigroups

within this class. One of the goals of this thesis has been to determine which properties are shared among all the quasigroups in a linear isotopy class, and which depend on the choices of A_1, A_2 and B. Our findings conclusively show that A_1 and A_2 do not influence the behavior of an isotopy class, but B do. In other words, the performance of the MQQ's in an isotopy class, with respect to a Gröbner basis attack, is invariant under changing A_1's and A_2's, but not B.

The matrix B is essential for enhancing the complexity of the MQQ. While the exact attributes of the B matrices that lead to weak keys are still unknown, we have found that it is largely responsible for the number of quadratic terms appearing in the MQQ's. This is an important complexity metric for Gröbner basis algorithms. The designers of MQQ-SIG had already recognized the importance of this metric, as indicative by their requirements on the ranks of the $\mathbf{B}_{f_i}$ matrices, which represent the quadratic terms in the MQQ's. However, they did not relate them to the security against Gröbner basis attacks, but rather to the completely different threat of MinRank attacks. We have shown that these rank requirements actually are very important when defending against Gröbner attacks, and that B is the most critical factor for creating high rank systems. Additionally, we found that the requirements in the original specification could be eased, while still maintaining the same security. This lead to a simple heuristic algorithm for creating keys, that potentially can be 300-400 times faster than using the stricter requirements.

Finally, we investigated the effect of using the specially constructed matrix S as the basis for the linear and affine maps $\mathcal{S}$ and $\mathcal{S}'$, rather than completely random matrices. Our results suggest that this choice should be reconsidered. While the design in MQQ-SIG has a performance comparable (albeit lower) to that of a random matrix, its behavior is much less predictable, occasionally producing very weak keys. With random matrices on the other hand, this never occurred.

To verify the authenticity of all our findings we created a modified key generation algorithm, adjusted to account for the results obtained on the individual parameters, and compared its performance to the original MQQ-SIG key generation. The results from these full system tests, support the adjustments we have recommended for the key generation algorithm. With our proposed changes, the system performs better on average, in addition to being very much more consistent. This last point is important, since it demonstrates that it is possible to avoid generating weak keys

in the MQQ-SIG system.

However, it is important to emphasize that the focus of this thesis has solely been on generic attacks based on Gröbner basis computations. We have not made any attempt to explicitly exploit the internal structure of MQQ-SIG, in particular we have not directly attacked the algebraic properties of quasigroups underlying the central map $\mathcal{P}'$. Withstanding generic attacks, while important, is only a minimal requirement. Much more cryptanalysis is needed to build enough confidence in the system to be trusted for real-world use. This is clearly the most important topic for further research on MQQ-SIG in particular, and the application of MQQ's to cryptography in general.

Another interesting area to explore, is how the quasigroup design can be used to construct a secure encryption scheme. Historically, it has proved difficult to turn the $\mathcal{MQ}$-problem into an encryption scheme that is both efficient and secure. The original MQQ cipher being a case in point. Investigating other ways of utilizing MQQ's can be a fruitful research direction.

Finally, key sizes is an important factor to consider. While MQQ-SIG is very fast, its keys sizes are also very large. This can be a real problem for its applicability on small devices with limited memory resources, such as embedded computers and microcontrollers. There is already beginning research into creating MQQ's over GF(p^k), as opposed to just GF(2), which can lead to large reductions in the public key size. Still, much is unknown about how these systems can be created efficiently.

References

[ACFP12] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, and Ludovic Perret. On the relation between the MXL family of algorithms and Gröbner basis algorithms. *Journal of Symbolic Computation*, 47:926–941, 2012. in press.

[Adl79] Leonard Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, SFCS '79, pages 55–60, Washington, DC, USA, 1979. IEEE Computer Society.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, STOC '96, pages 99–108, New York, NY, USA, 1996. ACM.

[BBD08] Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen. *Post Quantum Cryptography*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[BBG07] Côme Berbain, Olivier Billet, and Henri Gilbert. Efficient implementations of multivariate quadratic systems. In Eli Biham and Amr Youssef, editors, *Selected Areas in Cryptography*, volume 4356 of *Lecture Notes in Computer Science*, pages 174–187. Springer Berlin / Heidelberg, 2007.

[BCFS] Wieb Bosma, John J. Cannon, Claus Fieker, and Allan Steel. *Handbook of Magma functions, Edition 2.17-3 (2010), 5117 pages.*

[BFS99] Jonathan F. Buss, Gudmund S. Frandsen, and Jeffery O. Shallit. The computational complexity of some problems of linear algebra. *J. Comput. Syst. Sci.*, 58(3):572–596, June 1999.

[BFS03] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. Complexity of Gröbner basis computation for semi-regular overdetermined sequences over GF(2) with solutions in GF(2). Research Report 5049, Inria, December 2003. 19 pages.

[BL] Daniel J. Bernstein and Tanja Lange. eBACS: ECRYPT benchmarking of cryptographic systems. Accessed 16 may 2012. http://bench.cr.yp.to.

[Bon99] Dan Boneh. Twenty years of attacks on the RSA cryptosystem. *NOTICES OF THE AMS*, 46:203–213, 1999.

[Buc65] Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal.* PhD thesis, University of Innsbruck, 1965.

[CGM+02] Nicolas Courtois, Louis Goubin, Willi Meier, Jean-Daniel Tacier, and CP Crypto Lab. Solving underdefined systems of multivariate quadratic equations. In *Proceedings of Public Key Cryptography 2002, LNCS 2274*, pages 211–227. Springer-Verlag, 2002.

[Chr09] Adam Christov. Quasigroup based cryptography. Master's thesis, Charles University, Prague, 2009.

[CKG10] Yanling Chen, Svein J. Knapskog, and Danilo Gligoroski. Multivariate quadratic quasigroups (MQQ): Construction, bounds and complexity. In *Inscrypt, 6th China International Conference on Information Security and Cryptology.* Science Press of China, 2010.

[CKPS00] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate

polynomial equations. In *Proceedings of the 19th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'00, pages 392–407, Berlin, Heidelberg, 2000. Springer-Verlag.

[Cop97] Don Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. Cryptology*, 10:233–260, 1997.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[DS05] Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. In John Ioannidis, Angelos Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security*, volume 3531 of *Lecture Notes in Computer Science*, pages 317–366. Springer Berlin Heidelberg, 2005.

[DWY07] Jintai Ding, Christopher Wolf, and Bo-Yin Yang. ℓ-invertible cycles for multivariate quadratic public key cryptography. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography – PKC 2007*, volume 4450 of *Lecture Notes in Computer Science*, pages 266–281. Springer Berlin Heidelberg, 2007.

[Fau99] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, June 1999.

[Fau02] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, ISSAC '02, pages 75–83, New York, NY, USA, 2002. ACM.

[FJ03] Jean-Charles Faugère and Antoine Joux. Algebraic cryptanalysis of hidden field equation (HFE) cryptosystems using Gröbner bases. In *In Advances in Cryptology — CRYPTO 2003*, pages 44–60. Springer, 2003.

[FLDVP08] Jean-Charles Faugère, Françoise Levy-Dit-Vehel, and Ludovic Perret. Cryptanalysis of minrank. In *Proceedings of the 28th Annual conference on Cryptology: Advances in Cryptology*, CRYPTO 2008, pages 280–296, Berlin, Heidelberg, 2008. Springer-Verlag.

[FØPG10] Jean-Charles Faugère, Rune Steinsmo Ødegård, Ludovic Perret, and Danilo Gligoroski. Analysis of the MQQ public key cryptosystem. In Swee-Huay Heng, Rebecca N. Wright, and Bok-Min Goi, editors, *CANS*, volume 6467 of *Lecture Notes in Computer Science*, pages 169–183. Springer, 2010.

[FY79] Aviezri S. Fraenkel and Yaacov Yesha. Complexity of problems in games, graphs and algebraic equations. *Discrete Applied Mathematics*, 1(1–2):15 – 30, 1979.

[GC00] Louis Goubin and Nicolas Courtois. Cryptanalysis of the ttm cryptosystem. In *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, pages 44–57, 2000.

[GKSS08] Carla P. Gomes, Henry Kautz, Ashish Sabharwal, and Bart Selman. Chapter 2 satisfiability solvers. In Vladimir Lifschitz Frank van Harmelen and Bruce Porter, editors, *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 89 – 134. Elsevier, 2008.

[GMK08a] Danilo Gligoroski, Smile Markovski, and Svein J. Knapskog. Public key block cipher based on Multivariate Quadratic Quasigroups. *IACR Cryptology ePrint Archive*, 2008:320, 2008.

[GMK08b] Danilo Gligoroski, Smile Markovski, and Svein Johan Knapskog. Multivariate quadratic trapdoor functions based on multivariate quadratic quasigroups. In *Proceedings of the American Conference on Applied Mathematics*, MATH'08, pages 44–49, Stevens Point, Wisconsin, USA, 2008. World Scientific and Engineering Academy and Society (WSEAS).

[GØJ+11] Danilo Gligoroski, Rune Steinsmo Ødegård, Rune Erlend Jensen, Ludovic Perret, Jean-Charles Faugére, Svein Johan Knapskog, and Smile Markovski. MQQ-SIG, an ultra-fast and provably CMA resistant digital signature scheme. In *Proceedings of the 3rd International Conference on Trusted Systems, INTRUST*, 2011.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, New York, NY, USA, 2001.

[Gor93] Daniel M. Gordon. Discrete logarithms in gf(p) using the number field sieve. *SIAM J. Discrete Math*, 6:124–138, 1993.

[HHGPW10] Jeff Hoffstein, Nick Howgrave-Graham, Jill Pipher, and William Whyte. Practical lattice-based cryptography : NTRUEncrypt and NTRUSign. *The LLL algorithm: Surveys and Applications*, pages 1–42, 2010.

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, August 2001.

[KPG99] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced oil and vinegar signature schemes. In *IN ADVANCES IN CRYPTOLOGY — EUROCRYPT 1999*, pages 206–222. Springer, 1999.

[KR08] Martin Kreuzer and Lorenzo Robbiano. *Computational Commutative Algebra 1*. Springer Publishing Company, Incorporated, 2008.

[Len00] Arjen K. Lenstra. Integer factoring. *Design, Codes and Cryptography*, 19(2-3):101–128, March 2000.

[Mao03] Wenbo Mao. *Modern Cryptography: Theory and Practice*. Prentice Hall Professional Technical Reference, 2003.

[McE78] Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. Technical report, Jet Propulsion Lab Deep Space Network Progress report, 1978.

[MDBW09] Mohamed Saied Emam Mohamed, Jintai Ding, Johannes Buchmann, and Fabian Werner. Algebraic attack on the MQQ public key cryptosystem. In *Proceedings of the 8th International Conference on Cryptology and Network Security*, CANS '09, pages 392–401, Berlin, Heidelberg, 2009. Springer-Verlag.

[Mer89] Ralph C. Merkle. A certified digital signature. In *Proceedings on Advances in cryptology*, CRYPTO '89, pages 218–238, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[MH78] Ralph C. Merkle and Martin E. Hellman. Hiding information and signatures in trapdoor knapsacks. *Information Theory, IEEE Transactions on*, 24(5):525 – 530, sep 1978.

[MI88] Tsutomu Matsumoto and Hideki Imai. Public quadratic polynomial-tuples for efficient signature-verification and message-encryption. In *Lecture Notes in Computer Science on Advances in Cryptology-EUROCRYPT'88*, pages 419–453, New York, NY, USA, 1988. Springer-Verlag New York, Inc.

[MMDB08] Mohamed Saied Emam Mohamed, Wael Said Abd Elmageed Mohamed, Jintai Ding, and Johannes Buchmann. MXL2: Solving polynomial equations over GF(2) using an improved mutant strategy. In Johannes Buchmann and Jintai Ding, editors, *PQCrypto*, volume 5299 of *Lecture Notes in Computer Science*, pages 203–215. Springer, 2008.

[Moh99] Tzuong-Tsieng Moh. A fast public key system with signature and master key functions. In *CrypTEC'99 (Proc. International Workshop on Cryptographic Techniques & E-commerce)*. City University of Hong Kong Press, July 1999.

[Odl99] Andrew Odlyzko. Discrete logarithms: the past and the future. *Designs, Codes, and Cryptography*, 19:129–145, 1999.

[Pat96] Jacques Patarin. Hidden fields equations (HFE) and isomorphisms of polynomials (IP): Two new families of asymmetric algorithms. In *Proceedings of the 15th annual international conference on Theory*

and application of cryptographic techniques, EUROCRYPT'96, pages 33–48, Berlin, Heidelberg, 1996. Springer-Verlag.

[Pat00] Jacques Patarin. Cryptanalysis of the matsumoto and imai public key scheme of eurocrypt'98. *Designs, Codes and Cryptography*, 20:175–209, 2000.

[PG97] Jacques Patarin and Louis Goubin. Trapdoor one-way permutations and multivariate polynomials. In *Proc. of ICICS'97, LNCS 1334*, pages 356–368. Springer, 1997.

[PH78] Stephen Pohlig and Martin E. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1):106–110, 1978.

[Pol78] John M. Pollard. Monte Carlo methods of index computation (mod p). *Mathematics of Computation*, 32(143):918–924, 1978.

[RS99] Ronald L. Rivest and Robert D. Silverman. Are 'strong' primes needed for RSA? In *The 1997 RSA Laboratories Seminar Series, Seminars Proceedings*, 1999.

[RSA78] Ron R. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[SCG12] Simona Samardjiska, Yanling Chen, and Danilo Gligoroski. Algorithms for construction of multivariate quadratic quasigroups (MQQs) and their parastrophe operations in arbitrary galois fields. *Journal of Information Assurance and Security*, 7, 2012. (To appear).

[Sha71] Daniel Shanks. Class number, a theory of factorization, and genera. In *1969 Number Theory Institute (Proc. Sympos. Pure Math., Vol. XX, State Univ. New York, Stony Brook, N.Y., 1969)*, pages 415–440. Providence, R.I., 1971.

[Sha82] Adi Shamir. A polynomial time algorithm for breaking the basic merkle-hellman cryptosystem. In *Foundations of Computer Science,*

1982. SFCS '08. 23rd Annual Symposium on, pages 145 –152, nov. 1982.

[Sha93] Adi Shamir. Efficient signature schemes based on birational permutations. In *Proceedings of CRYPTO'93, number 773 in LNCS*, pages 1–12. Springer-Verlag, 1993.

[Sho97a] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, October 1997.

[Sho97b] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *Proceedings of the 16th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'97, pages 256–266, Berlin, Heidelberg, 1997. Springer-Verlag.

[Smi07] Jonathan D.H. Smith. *An Introduction to Quasigroups and Their Representations*. Studies in Advanced Mathematics. Chapman & Hall/CRC, 2007.

[Wie90] Michael J. Wiener. Cryptanalysis of short RSA secret exponents. *IEEE Transactions on Information Theory*, 36:553–558, 1990.

[WP05a] Christopher Wolf and Bart Preneel. Equivalent keys in HFE, C*, and variations. In *In Mycrypt 2005, Lecture Notes in Computer Science LNCS 3715, Ed Dawson, Serge Vaudenay (Eds*, pages 33–49. Springer-Verlag, 2005.

[WP05b] Christopher Wolf and Bart Preneel. Taxonomy of public key schemes based on the problem of multivariate quadratic equations. Cryptology ePrint Archive, Report 2005/077, 2005. http://eprint.iacr.org/.

[YC05] Bo-Yin Yang and Jiun-Ming Chen. Building secure tame-like multivariate public-key cryptosystems: the new TTS. In *Proceedings of the 10th Australasian conference on Information Security and Privacy*, ACISP'05, pages 518–531, Berlin, Heidelberg, 2005. Springer-Verlag.