Proceedings of the Second International Conference on Symbolic Computation and Cryptography

Carlos Cid and Jean-Charles Faugère (Eds.)

23 – 25 June 2010,
Royal Holloway, University of London, Egham, UK

Programme Chairs

Carlos Cid	Royal Holloway, University of London, UK
Jean-Charles Faugère	UPMC-INRIA, France

Programme Committee

Daniel Bernstein	University of Illinois at Chicago, USA
Olivier Billet	Orange Labs, France
Claude Carlet	University of Paris 8, France
Pierre-Alain Fouque	ENS - Paris, France
Joachim von zur Gathen	Universität Paderborn, Germany
Pierrick Gaudry	CNRS, France
Jaime Gutierrez	University of Cantabria, Spain
Antoine Joux	Université de Versailles Saint-Quentin-en-Yvelines, France
Martin Kreuzer	Universität Passau, Germany
Dongdai Lin	Institute of Software of Chinese Academy of Sciences, China
Alexander May	Ruhr-Universität Bochum, Germany
Ayoub Otmani	GREYC-Ensicaen & University of Caen & INRIA, France
Ludovic Perret	LIP6-UPMC Univ Paris 6 & INRIA, France
Igor Shparlinski	Macquarie University, Australia
Boaz Tsaban	Bar-Ilan University, Israel
Maria Isabel González Vasco	Universidad Rey Juan Carlos, Spain

Algebraic Attacks Using SAT-Solvers

Philipp Jovanovic and Martin Kreuzer

Fakultät für Informatik und Mathematik
Universität Passau
D-94030 Passau, Germany

Abstract. Algebraic attacks lead to the task of solving polynomial systems over $\mathbb{F}_2$. We study recent suggestions of using SAT-solvers for this task. In particular, we develop several strategies for converting the polynomial system to a set of CNF clauses. This generalizes the approach in [4]. Moreover, we provide a novel way of transforming a system over $\mathbb{F}_{2^e}$ to a (larger) system over $\mathbb{F}_2$. Finally, the efficiency of these methods is examined using standard examples such as CTC, DES, and Small Scale AES.

Key words: algebraic cryptanalysis, SAT solver, AES, polynomial system solving

1 Introduction

The basic idea of algebraic cryptanalysis is to convert the problem of breaking a cypher to the problem of solving a system of polynomial equations over a finite field, usually a field of characteristic 2. A large number of different approaches has been developed to tackle such polynomial systems (for an overview, see [12]).

In this note we examine a recent suggestion, namely to convert the system to a set of propositional logic clauses and then to use a SAT-solver. In [8] this was successfully applied to attack 6 rounds of DES. The first study of efficient methods for converting boolean polynomial systems to CNF clauses was presented in [4] where the following procedure was suggested:

(1) Linearise the system by introducing a new indeterminate for each term in the support of one of the polynomials.
(2) Having written a polynomial as a sum of indeterminates, introduce new indeterminates to cut it after a certain number of terms. (This number is called the *cutting number*.)
(3) Convert the reduced sums into their logical equivalents using a XOR-CNF conversion.

Later this study was extended slightly in [5], [3], and [17] but the procedure was basically unaltered. Our main topic, discussed in Section 2 of this paper, is to examine different *conversion strategies*, i.e. different ways to convert the polynomial system into a satisfiability problem. The crucial point is that the linearisation phase (1) usually produces way too many new indeterminates. Our goal will be to substitute not single terms, but term combinations, in order to save indeterminates and clauses in the CNF output.

For certain cryptosystems such as AES (see [9]) or its small scale variants (see [6]), the polynomial systems arising from an algebraic attack are naturally defined over a finite extension field $\mathbb{F}_{2^e}$ of $\mathbb{F}_2$, for instance over $\mathbb{F}_{16}$ or $\mathbb{F}_{256}$. While it is clear that one can convert a polynomial system over $\mathbb{F}_{2^e}$ to a polynomial system over $\mathbb{F}_2$ by introducing additional indeterminates, it is less clear what the best way is to do this such that the resulting system over $\mathbb{F}_2$ allows a good conversion to a SAT problem. An initial discussion of this question is contained in [2]. Although the algorithm we present in Section 3 is related to the one given there, our method seems to be easier to implement and to allow treatment of larger examples.

In the last section we report on some experiments and timings using the first author's implementation of our strategies in the ApCoCoA system (see [1]). By looking at the Courtois Toy Cipher (CTC), the Data Encryption Standard (DES), and Small Scale AES, we show that a suitably chosen conversion strategy can save a substantial amount of logical variables and clauses in the CNF output. The typical savings are in the order of 10% of the necessary logical variables and up to 25% in the size of the set of clauses. The main benefit is then a significant speed-up of the SAT-solvers which are applied to these sets of clauses. Here the gain can easily be half of the execution time or even more. We shall also see that a straightforward Gröbner basis approach to solving polynomial systems over $\mathbb{F}_2$ is slower by several orders of magnitude.

This paper is based on the first author's thesis [11]. Unless explicitly noted otherwise, we adhere to the definitions and notation of [13] and [14].

2 Converting Boolean Polynomials to CNF Clauses

In this section we let $\mathbb{F}_2$ be the field with two elements and $f \in \mathbb{F}_2[x_1, \dots, x_n]$ a polynomial. Usually f will be a *boolean polynomial*, i.e. all terms in the support of f will be squarefree, but this is not an essential hypothesis. Let $M = \{X_1, \dots, X_n\}$ be a set of *boolean variables* (atomic formulas), and let $\widehat{M}$ be the set of all (propositional) logical formulas that can be constructed from them, i.e. all formulas involving the operations $\neg$, $\wedge$, and $\vee$.

The following definition describes the relation between the zeros of a polynomial and the evaluation of a logical formula.

Definition 1. *Let $f \in \mathbb{F}_2[x_1, \dots, x_n]$ be a polynomial. A logical formula $F \in \widehat{M}$ is called a* **logical representation** *of f if $\varphi_a(F) = f(a_1, \dots, a_n) + 1$ for every $a = (a_1, \dots, a_n) \in \mathbb{F}_2^n$. Here φ_a denotes the boolean value of F at the tuple of boolean values a where $1 =$ `true` and $0 =$ `false`.*

The main effect of this definition is that boolean tuples at which F is satisfied correspond uniquely to zeros of f in $\mathbb{F}_2^n$. The following two lemmas contain useful building blocks for conversion strategies.

Lemma 2. *Let $f \in \mathbb{F}_2[x_1, \dots, x_n]$ be a polynomial, let $F \in \widehat{M}$ be a logical representation of f, let y be a further indeterminate, and let Y be a further boolean variable. Then $G = (\neg F \Leftrightarrow Y)$ is a logical representation of the polynomial $g = f + y$.*

Proof. Let $\bar{a} = (a_1, \dots, a_n, b) \in \mathbb{F}_2^{n+1}$. We distinguish two cases.

(1) If $b = 1$ then $g(\bar{a}) = f(a) + 1 = \varphi_a(F)$. Since $\varphi_b(Y) = 1$, we get $\varphi_{\bar{a}}(\neg F \Leftrightarrow Y) = \varphi_a(\neg F) = \varphi_a(F) + 1$.
(2) If $b = 0$ then $g(\bar{a}) = f(a) = \varphi_a(F) + 1$ and $\varphi_b(Y) = 0$ implies $\varphi_{\bar{a}}(\neg F \Longleftrightarrow Y) = \varphi_a(F)$.

In both cases we find $\varphi_{\bar{a}}(\neg F \Longleftrightarrow Y) = g(\bar{a}) + 1$, as claimed. $\square$

The preceding lemma is the work horse for the standard conversion algorithm. The next result extends it in a useful way.

Lemma 3. *Let $f \in \mathbb{F}_2[x_1, \dots, x_n, y]$ be a polynomial of the form $f = \ell_1 \cdots \ell_s + y$ where $1 \leq s \leq n$ and $\ell_i \in \{x_i,\, x_i + 1\}$ for $i = 1, \dots, s$. We define formulas $L_i = X_i$ if $\ell_i = x_i$ and $L_i = \neg X_i$ if $\ell_i = x_i + 1$. Then*

$$F = (\neg Y \vee L_1) \wedge \ldots \wedge (\neg Y \vee L_s) \wedge (Y \vee \neg L_1 \vee \ldots \vee \neg L_s)$$

is a logical representation of f. Notice that F is in conjunctive normal form (CNF) *and has $s + 1$ clauses.*

Proof. Let $a = (a_1, \dots, a_n, b) \in \mathbb{F}_2^{n+1}$ We will show $\varphi_a(F) = f(a) + 1$ by induction on s. In the case $s = 1$ we have $f = x_1 + y + c$ where $c \in \{0, 1\}$ and $F = (\neg Y \vee L_1) \wedge (Y \vee \neg L_1)$ where $L_1 = X_1$ if $c = 0$ and $L_1 = \neg X_1$ if $c = 1$. The claim $\varphi_a(F) = f(a) + 1$ follows easily with the help of a truth table.

Now we prove the inductive step, assuming that the claim has been shown for $s - 1$ factors ℓ_i, i.e. for $f' = \ell_1 \cdots \ell_{s-1}$ and the corresponding formula F'. To begin with, we assume that $\ell_s = x_s$ and distinguish two sub-cases.

(1) If $a_s = 0$, we have $\varphi_a(F) = \varphi_a(\neg Y \vee L_1) \wedge \ldots \wedge (\neg Y \vee L_{s-1}) \wedge \neg Y = \varphi_b(\neg Y)$ and $f(a) = b$. This shows $\varphi_a(F) = f(a) + 1$.
(2) If $a_s = 1$, we have $f(a) = f'(a)$. Using $\varphi_a(L_s) = 1$, we obtain

$$\varphi_a(F) = \varphi_a(\neg Y \vee L_1) \wedge \ldots \wedge (\neg Y \vee L_{s-1}) \wedge (Y \vee \neg L_1 \vee \ldots \vee \neg L_{s-1}) = \varphi_a(F')$$

Hence the inductive hypothesis yields $\varphi_a(F) = \varphi_a(F') = f'(a) + 1 = f(a) + 1$.

In the case $\ell_s = x_s + 1$, the proof proceeds in exactly the same way. $\square$

Based on these lemmas, we can define three elementary strategies for converting systems of (quadratic) polynomials over $\mathbb{F}_2$ into linear systems and a set of CNF clauses.

Definition 4. *Let $f \in \mathbb{F}_2[x_1, \dots, x_n]$ be a polynomial.*

(1) For each non-linear term t in the support of f, introduce a new indeterminate y and a new boolean variable Y. Substitute y for t in f and append the clauses corresponding to $t+y$ in Lemma 3 to the set of clauses. This is called the **standard strategy (SS)**.

(2) Assuming $\deg(f) = 2$*, try to find combinations $x_ix_j + x_i$ in the support of f. Introduce a new indeterminate y and a new boolean variable Y. Replace $x_ix_j + x_i$ in f by y and append the clauses corresponding to $x_i(x_j + 1) + y$ in Lemma 3 to the set of clauses. This is called the* **linear partner strategy (LPS)**.

(3) Assuming $\deg(f) = 2$*, try to find combinations $x_ix_j + x_i + x_j + 1$ in the support of f. Introduce a new indeterminate y and a new boolean variable Y. Replace $x_ix_j + x_i + x_j + 1$ in f by y and append the clauses corresponding to $(x_i + 1)(x_j + 1) + y$ in Lemma 3 to the set of clauses. This is called the* **double partner strategy (DPS)**.

Let compare the effect of these strategies in a simple example.

Example 5. Consider the polynomial $f = x_1x_2 + x_1x_3 + x_2x_3 + x_1 + x_2 + 1$ in $\mathbb{F}_2[x_1, x_2, x_3]$. The following table lists the number of additional logical variables (#v) and clauses (#c) each strategy produces during the conversion of this polynomial to a set of CNF clauses.

strategy	SS	LPS	DPS
# v	4	3	3
# c	25	17	13

Even better results can be achieved for quadratic and cubic terms by applying the following two propositions.

Proposition 6 (Quadratic Partner Substitution).
Let $f = x_ix_j + x_ix_k + y \in \mathbb{F}_2[x_1, \dots, x_n, y]$ be a polynomial such that i, j, k are pairwise distinct. Then

$$\begin{aligned} F = \; & (X_i \vee \neg Y) \wedge (X_j \vee X_k \vee \neg Y) \wedge (\neg X_j \vee \neg X_k \vee \neg Y) \wedge \\ & (\neg X_i \vee \neg X_j \vee X_k \vee Y) \wedge (\neg X_i \vee X_j \vee \neg X_k \vee Y) \end{aligned}$$

is a logical representation of f.

Proof. Using a truth table it is easy to check that the polynomial $g = x_ix_j + x_ix_k \in \mathbb{F}_2[x_1, \dots, x_n]$ has the logical representation

$$G = (\neg X_i \vee \neg X_j \vee X_k) \wedge (\neg X_i \vee X_j \vee \neg X_k)$$

Now Lemma 2 implies that the formula $F = \neg G \Leftrightarrow Y$ represents f, and after applying some simplifying equivalences we get the claimed formula. □

It is straightforward to formulate a conversion strategy, called the **quadratic partner strategy (QPS)**, for polynomials of degree two based on this proposition. Let us see how this strategy performs in the setting of Example 5.

Example 7. Let $f = x_1x_2 + x_1x_3 + x_2x_3 + x_1 + x_2 + 1 \in \mathbb{F}_2[x_1, x_2, x_3]$. Then QPS introduces 2 new logical variables and produces 16 additional clauses. Although the number of clauses is higher than for DPS, the lower number of new indeterminates is usually more important and provides superior timings.

For cubic terms, e.g. the ones appearing in the equations representing DES, the following substitutions can be used.

Proposition 8 (Cubic Partner Substitution).
Let $f = x_ix_jx_k + x_ix_jx_l + y \in \mathbb{F}_2[x_1, \ldots, x_n, y]$, where i, j, k, l are pairwise distinct. Then

$$\begin{aligned} F = {} & (X_i \vee \neg Y) \wedge (X_j \vee \neg Y) \wedge (X_k \vee X_l \vee \neg Y) \wedge (\neg X_k \vee \neg X_l \vee \neg Y) \wedge \\ & (\neg X_i \vee \neg X_j \vee \neg X_k \vee X_l \vee Y) \wedge (\neg X_i \vee \neg X_j \vee \neg X_k \vee \neg X_l \vee Y) \end{aligned}$$

is a logical representation for f.

Proof. Using a truth table it is easy to check that the polynomial $g = x_ix_jx_k + x_ix_jx_l \in \mathbb{F}_2[x_1, \ldots, x_n]$ has the logical representation

$$G = (\neg X_i \vee \neg X_j \vee \neg X_k \vee X_l) \wedge (\neg X_i \vee \neg X_j \vee X_k \vee \neg X_l)$$

Now Lemma 2 yields the representation $F = \neg G \Leftrightarrow Y$ for f, and straightforward simplification produces the desired result. □

By inserting this substitution method into the conversion algorithm, we get the **cubic partner strategy (CPS)**. In Section 4 we shall see the savings in clauses, indeterminates, and execution time one can achieve by applying this strategy to DES. For cubic terms, it is also possible to pair them if they have just one indeterminate in common. However, this strategy apparently does not result in useful speed-ups and is omitted.

To end this section, we combine the choice of a substitution strategy with the other steps of the conversion algorithm and spell out the version which we implemented and used for the applications and timings in Section 4.

Proposition 9 (Boolean Polynomial System Conversion).
Let $f_1, \ldots, f_m \in \mathbb{F}_2[x_1, \ldots, x_n]$, and let $\ell \geq 3$ be the desired cutting number. Consider the following sequence of instructions.

C1. *Let $G = \emptyset$. Perform the following steps* **C2-C5** *for $i = 1, \ldots, m$.*
C2. *Repeat the following step* **C3** *until no polynomial g can be found anymore.*
C3. *Find a subset of* $\operatorname{Supp}(f_i)$ *which defines a polynomial g of the type required by the chosen conversion strategy. Introduce a new indeterminate y_j, replace f_i by $f_i - g + y_j$, and append $g + y_j$ to G.*
C4. *Perform the following step* **C5** *until $\# \operatorname{Supp}(f_i) \leq \ell$. Then append f_i to G.*
C5. *If $\# \operatorname{Supp}(f_i) > \ell$ then introduce a new indeterminate y_j, let g be the sum of the first $\ell - 1$ terms of f_i, replace f_i by $f_i - g + y_j$, and append $g + y_j$ to G.*

C6. *For each polynomial in G, compute a logical representation in CNF. Return the set of all clauses K of all these logical representations.*

This is an algorithm which computes (in polynomial time) a set of CNF clauses K such that the boolean tuples satisfying K are in 1-1 correspondence with the solutions of the polynomial system $f_1 = \cdots = f_m = 0$.

Proof. It is clear that steps **C2-C3** correspond to the linearisation part (1) of the procedure given in the introduction, and that steps **C4-C5** are an explicit version of the cutting part (2) of that procedure. Moreover, step **C6** is based on Lemma 2, Lemma 3, Prop. 6, or Prop. 8 for the polynomials $g + y_j$ from step **C3**, and on the standard XOR-CNF conversion for the linear polynomials from steps **C4-C5**. The claim follows easily from these observations. □

3 Converting Char 2 Polynomials to Boolean Polynomials

In the following we let $e > 0$, and we suppose that we are given polynomials $f_1, \dots, f_m \in \mathbb{F}_{2^e}[x_1, \dots, x_n]$. Our goal is to use SAT-solvers to solve the system $f_1(x_1, \dots, x_n) = \cdots = f_m(x_1, \dots, x_n) = 0$ over the field $\mathbb{F}_{2^e}$. For this purpose we represent the field $\mathbb{F}_{2^e}$ in the form $\mathbb{F}_{2^e} \cong \mathbb{F}_2[x]/\langle g\rangle$ with an irreducible, unitary polynomial g of degree e.

Notice that every element a of $\mathbb{F}_{2^e}$ has a unique representation $a = a_1 + a_2\bar{x} + a_3\bar{x}^2 + \cdots + a_e\bar{x}^{e-1}$ with $a_i \in \{0, 1\}$. Here $\bar{x}$ denotes the residue class of x in $\mathbb{F}_{2^e}$. Let ε be the homomorphism of $\mathbb{F}_2$-algebras

$$\varepsilon : (\mathbb{F}_2[x]/\langle g\rangle)[x_1, \dots, x_n] \longrightarrow (\mathbb{F}_2[x]/\langle g\rangle)[y_1, \dots, y_{en}]$$

given by $x_i \mapsto y_{(i-1)e+1} + y_{(i-1)e+2} \cdot \bar{x} + \cdots + y_{ie} \cdot \bar{x}^{e-1}$ for $i = 1, \dots, n$.

Proposition 10 (Base Field Transformation).
In the above setting, consider the following sequence of instructions.

F1. *Perform the following steps* **F2-F5** *for $i = 1, \dots, m$.*
F2. *For each term $t \in \mathrm{Supp}(f_i)$ compute a representative t'' for $\varepsilon(t)$ using the following steps* **F3-F4**. *Recombine the results to get a representative for $\varepsilon(f_i)$.*
F3. *Apply the substitutions $x_j \mapsto y_{(j-1)e+1} + y_{(j-1)e+2} \cdot \bar{x} + \cdots + y_{je} \cdot \bar{x}^{e-1}$ and get a polynomial t'.*
F4. *Compute $t'' = \mathrm{NR}_{Q\cup\{g\}}(t')$. Here $Q = \{y_k^2 - y_k \mid k = 1, \dots, ne\}$ is the set of* field equations *of $\mathbb{F}_2$ and the* normal remainder NR *is computed by the Division Algorithm (see [13], Sect. 1.6).*
F5. *Write $\varepsilon(f_i) = h_{i1} + h_{i2}\bar{x} + \cdots + h_{ie}\bar{x}^{e-1}$ with $h_{ij} \in \mathbb{F}_2[y_1, \dots, y_{ne}]$.*
F6. *Return the set $H = \{h_{ij}\}$.*

This is an algorithm which computes a set of polynomials H in $\mathbb{F}_2[y_1, \dots, y_{en}]$ such that the $\mathbb{F}_2$-rational common zeros of H correspond 1-1 to the F_{2^e}-rational solutions of $f_1 = \cdots = f_m = 0$.

Proof. Using the isomorphism $\mathbb{F}_{2^e} \cong \mathbb{F}_2[x]/\langle g\rangle$ we represent the elements of $\mathbb{F}_{2^e}$ uniquely as polynomials of degree $\le e-1$ in the indeterminate x. Let $a = (a_1, \dots, a_n) \in \mathbb{F}_{2^e}^n$ be a solution of the system $f_1 = \cdots = f_m = 0$. For $k = 1, \dots, n$, we write $a_k = c_{k1} + c_{k2}\bar{x} + \cdots + c_{ke}\bar{x}^{e-1}$ with $c_{k\ell} \in \{0, 1\}$.

By the definition of ε, we see that $(c_{11}, \dots, c_{ne})$ is a common zero of the polynomials $\{\varepsilon(f_1), \dots, \varepsilon(f_m)\}$. Since $\{1, \bar{x}, \dots, \bar{x}^{e-1}\}$ is a basis of the $\mathbb{F}_2[y_1, \dots, y_{ne}]$-module $\mathbb{F}_2[\bar{x}][y_1, \dots, y_{ne}]$, the tuple $(c_{11}, \dots, c_{1e})$ is actually a common zero of all coefficient polynomials h_{ij} of each $\varepsilon(f_i)$.

In the same way it follows that, conversely, every common zero $(c_{11}, \dots, c_{ne}) \in \mathbb{F}_2^{ne}$ of H yields a solution $(a_1, \dots, a_n)$ of the given polynomial system over $\mathbb{F}_{2^e}$ via $a_k = c_{k1} + c_{k2}\bar{x} + \cdots + c_{ke}\bar{x}^{e-1}$. $\square$

In the computations below we used the representations $\mathbb{F}_{16} = \mathbb{F}_2[x]/\langle x^4 + x + 1\rangle$ for Small Scale AES and $\mathbb{F}_{256} = \mathbb{F}_2[x]/\langle x^8 + x^4 + x^3 + x + 1\rangle$ for the full AES. They correspond to the specifications in [6] and [9], respectively.

4 Applications and Timings

In this section we report on some experiments with the new conversion strategies and compare them to the standard strategy. Moreover, we compare some of the timings we obtained to the straightforward Gröbner basis approach. For the cryptosystems under consideration, we used the ApCoCoA implementations by J. Limbeck (see [15]).

As in [4], the output of the conversion algorithms are files in the DIMACS format which is used by most SAT-solvers. The only exception is the system CryptoMiniSat which uses a certain XOR-CNF file format (see [17]). The conversion algorithm generally used the cutting number 4 (exceptions are indicated). The timings were obtained by running MiniSat 2.1 (see [10]) resp. the XOR extension of CryptoMiniSat (see [16]) on a 2.66 GHz Quadcore Xeon PC with 8 GB RAM. The timings for the conversion of the polynomial system to a set of CNF clauses were ignored, since the conversion was not implemented efficiently and should be seen as a preprocessing step. Finally, we note that the timings represent averages of 20 runs of the SAT solvers, for each of which we have randomly permuted the set of input clauses. The reason for this procedure is that SAT solvers are randomized algorithms which rely heavily on heuristical methods.

4.1 The Courtois Toy Cipher (CTC)

This artificial cryptosystem was described in [7]. Its complexity is configurable. We denote the system having n encryption rounds and b parallel S-boxes by CTC(n,b). In the following table we collect the savings in logical variables ($\#v$) and clauses ($\#c$) we obtain by using different conversion strategies. The S-boxes were modelled using the full set of 14 equations each.

system	$(\#v_{SS}, \#c_{SS})$	$(\#v_{LPS}, \#c_{LPS})$	$(\#v_{DPS}, \#c_{DPS})$	$(\#v_{QPS}, \#c_{QPS})$
CTC(3,3)	(361, 2250)	(352, 1908)	(334, 1796)	(352, 2246)
CTC(4,4)	(617, 4017)	(601, 3409)	(569, 3153)	(617, 3953)
CTC(5,5)	(956, 6266)	(931, 5316)	(881, 4916)	(956, 6116)
CTC(6,6)	(1369, 8989)	(1333, 7621)	(1261, 7045)	(1369, 8845)

Thus the LPS and QPS conversions do not appear to provide substantial improvements over the standard strategy, but DPS reduces the input for the SAT-solver by about 8% variables and 22% clauses. Let us see whether this results in a meaningful speed-up. To get significant execution times with MiniSat, we consider the system CTC(6,6). We restrict ourselves to the optimized number of 7 equations per S-box, i.e. we are solving a system of 612 equations in 468 indeterminates over $\mathbb{F}_2$, and we use MiniSat (time t_M in seconds) and the XOR version of CryptoMiniSat (time t_C in seconds).

strategy	# v	# c	t_M	t_C
SS	937	5065	27.5	39.3
LPS	865	4417	19.0	33.9
DPS	793	3841	24.3	43.5
QPS	937	5065	25.4	34.5

Thus LPS resulted in a 31% speed-up of MiniSat and in a 14% speed-up of CryptoMiniSat. The reduction by 15% variables and 24% clauses using DPS provided a 12% speed-up of MiniSat and no speed-up of CryptoMiniSat. Although QPS did not save any logical variables or clauses, it speeded up MiniSat by 8% and CryptoMiniSat by 12%. By combining these methods with other optimizations, significantly larger CTC examples can be solved.

4.2 The Data Encryption Standard (DES)

Next we examine the application of SAT-solvers to DES. By DES-n we denote the system of equations resulting from an algebraic attack at n rounds of DES. To model the S-boxes, we used optimized sets of 10 or 11 equations that we computed via the technique explained in [12]. Since the S-box equations are mostly composed of terms of degree 3, we compare SS conversion to CPS conversion. The optimal cutting number for SS turned out to be 4 or 5 in most cases.

In the following table we provide the number of polynomial indeterminates ($\#i$), the number of polynomial equations ($\#e$), the number of logical variables ($\#v$) and clauses ($\#c$) resulting from the SS and the CPS conversion, together with some timings of MiniSat ($t_{M,S}$ and $t_{M,CPS}$), as well as the XOR version of CryptoMiniSat (t_C). (The timings are in seconds, except where indicated.)

system	$\#i$	$\#e$	$(\#v_{SS}, \#c_{SS})$	$(\#v_{CPS}, \#c_{CPS})$	$t_{M,SS}$	t_C	$t_{M,CPS}$
DES-3	400	550	(5114, 34075)	(4583, 30175)	0.06	0.36	0.06
DES-4	512	712	(6797, 45428)	(6089, 40228)	390	179	310
DES-5	624	874	(8480, 56775)	(6205, 71361)	33h	96h	4.8h

From this table we see that, for DES-3 and DES-4, the CPS reduces the number of logical variables and clauses by about 11% each. For instance, for DES-4 this results in a 21% speed-up of MiniSat. In the case of DES-5 we used cutting length 6 for CPS and achieved a 27% reduction in the number of logical variables and an 85% speed-up.

4.3 Small Scale AES and Full AES

Let us briefly recall the arguments of possible configurations of the Small Scale AES cryptosystem presented in [6]. By AES(n,r,c,e) we denote the system such that

- $n \in \{1, \ldots, 10\}$ is the number of encryption rounds,
- $r \in \{1, 2, 4\}$ is the number of rows in the rectangular input arrangement,
- $c \in \{1, 2, 4\}$ is the number of columns in the rectangular input arrangement,
- $e \in \{4, 8\}$ is the bit size of a word.

The word size e indicates the field $\mathbb{F}_{2^e}$ over which the equations are defined, i.e. $e = 4$ corresponds to $\mathbb{F}_{16}$ and $e = 8$ to $\mathbb{F}_{256}$. By choosing the parameters $r = 4$, $c = 4$ and $w = 8$ one gets a block size of $4 \cdot 4 \cdot 8 = 128$ bits, and Small Scale AES becomes AES.

For the cutting number in the following tables, we used the number 4, since this turned out to be generally the best choice. Let us begin with a table which shows the savings in logical variables and clauses one can achieve by using the QPS conversion method instead of the standard strategy. Notice that AES yields linear polynomials or homogeneous polynomials of degree 2. Thus QPS is the only conversion strategy suitable for minimalizing the number of logical variables and clauses.

In the following table we list the number of indeterminates ($\#i$) and equations ($\#e$) of the original polynomial system, as well as the number of logical variables and clauses of its CNF conversion using the standard strategy ($\#v_{SS}, \#c_{SS}$) and the QPS conversion ($\#v_{QPS}, \#c_{QPS}$).

AES(n,r,c,w)	$\#i$	$\#e$	$(\#v_{SS}, \#c_{SS})$	$(\#v_{QPS}, \#c_{QPS})$
AES(9,1,1,4)	592	1184	(2905, 17769)	(2617, 16905)
AES(4,2,1,4)	544	1088	(3134, 20123)	(2878, 19355)
AES(2,2,2,4)	512	1024	(2663, 17611)	(2471, 17035)
AES(3,1,1,8)	832	1664	(11970, 83509)	(11010, 78469)
AES(1,2,2,8)	1152	2304	(14125, 101249)	(13165, 96209)
AES(2,2,2,8)	2048	4096	(32530, 236669)	(30610, 226589)
AES(1,4,4,8)	4352	8704	(52697, 383849)	(49497, 367049)

Although the QPS conversion reduces the logical indeterminates in the CNF output by only about 6-8% and the clauses by an even meagerer 4-5%, we will see that the speed-up for MiniSat can be substantial, e.g. about 27% for one round of full AES.

Our last table provides some timings for MiniSat with respect to the SS conversion set of clauses ($t_{M,SS}$), with respect to the QPS conversion set of clauses ($t_{M,QPS}$), and of the XOR version of CryptoMiniSat (t_C).

AES(n,r,c,w)	$t_{M,SS}$	t_C	$t_{M,QPS}$
AES(9,1,1,4)	0.08	0.02	0.07
AES(4,2,1,4)	1.31	0.58	0.50
AES(2,2,2,4)	0.83	0.62	0.71
AES(3,1,1,8)	73.6	114	55.7
AES(1,2,2,8)	112	131	61.7
AES(2,2,2,8)	(7079)	73h	(6767)
AES(1,4,4,8)	5072	15h	3690

Thus the QPS conversion usually yields a sizeable speed-up. Notice that, for small and medium size examples, also the XOR version of CryptoMiniSat provides competitive timings. The timings for AES(2,2,2,8) are based on cutting lengths 5 and 3, respectively, since these choices turned out to be significantly faster. The timings also depend on the chosen plaintext-ciphertext pairs. Above we give typical timings. In extreme cases the gain resulting from our strategies can be striking. For instance, for one plaintext-ciphertext pair in AES(3,1,1,8) we measured 222 sec. for the SS strategy and 0.86 sec. for the QPS strategy.

When we compare these timings to the Gröbner basis approach in [15], we see that SAT-solvers are vastly superior. Of the preceding 7 examples, only the first two finish without exceeding the 8 GB RAM limit. They take 596 sec. and 5381 sec. respectively, compared to fractions of a second for the SAT-solvers.

Finally, we note that the timings seem to depend on the cutting number in a rather subtle and unpredictable way. For instance, the best timing for one full round of AES was obtained by using the QPS conversion and a cutting number of 6. In this case, MiniSat was able to solve that huge set of clauses in 716 seconds, i.e. in less than 12 minutes. Clearly, the use of SAT-solvers in cryptanalysis opens up a wealth of new possibilities.

Acknowledgements. The authors are indebted to Jan Limbeck for the possibility of using his implementations of various cryptosystems in ApCoCoA (see [15]) and for useful advice. They also thank Stefan Schuster for valuable discussions and help with the implementations underlying Sect. 4.

References

1. ApCoCoA team: ApCoCoA: Applied Computations in Commutative Algebra. Available at `http://www.apcocoa.org`
2. Bard, G.: On the rapid solution of systems of polynomial equations over low-degree extension fields of GF(2) via SAT-solvers. In: 8th Central European Conf. on Cryptography (2008)
3. Bard, G.: Algebraic Cryptanalysis. Springer Verlag (2009)

4. Bard, G., Courtois, N., Jefferson, C.: Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF(2) via SAT-solvers. Cryptology ePrint Archive 2007(24) (2007)
5. Chen, B.: Strategies on algebraic attacks using SAT solvers. In: 9th Int. Conf. for Young Computer Scientists. IEEE Press (2008)
6. Cid, C., Murphy, S., Robshaw, M.: Small scale variants of the AES. In: Fast Software Encryption: 12th International Workshop. pp. 145–162. Springer Verlag, Heidelberg (2005)
7. Courtois, N.: How fast can be algebraic attacks on block ciphers. In: Biham, E., Handschuh, H., Lucks, S., Rijmen, V. (eds.) Symmetric Cryptography – Dagstuhl 2007. Dagstuhl Sem. Proc., vol. 7021. Available at `http://drops.dagstuhl.de/opus/volltexte/2007/1013`
8. Courtois, N., Bard, G.: Algebraic cryptanalysis of the data encryption standard. In: Galbraith, S. (ed.) IMA International Conference on Cryptography and Coding Theory. LNCS, vol. 4887, pp. 152–169. Springer Verlag (2007)
9. Daemen, J., Rijmen, V.: The Design of Rijndael. AES – The Advanced Encryption Standard. Springer Verlag, Berlin (2002)
10. Eèn, N., Sörensen, N.: Minisat. Available at `http://minisat.se`
11. Jovanovic, P.: Lösen polynomieller Gleichungssysteme über $\mathbb{F}_2$ mit Hilfe von SAT-Solvern. Universität Passau (2010)
12. Kreuzer, M.: Algebraic attacks galore! Groups - Complexity - Cryptology 1, 231–259 (2009)
13. Kreuzer, M., Robbiano, L.: Computational Commutative Algebra 1. Springer Verlag, Heidelberg (2000)
14. Kreuzer, M., Robbiano, L.: Computational Commutative Algebra 2. Springer Verlag, Heidelberg (2005)
15. Limbeck, J.: Implementation und Optimierung algebraischer Angriffe. Diploma thesis, Universität Passau (2008)
16. Soos, M., Nohl, K., Castelluccia, C.: CryptoMiniSat. Available at `http://planete.inrialpes.fr/∼soos/`
17. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: Kullmann, O. (ed.) Theory and Applications of Satisfiability Testing – SAT 2009. LNCS, vol. 5584. Springer Verlag (2009)

Cold Boot Key Recovery using Polynomial System Solving with Noise

Martin Albrecht* and Carlos Cid

Information Security Group,
Royal Holloway, University of London
Egham, Surrey TW20 0EX, United Kingdom
{M.R.Albrecht,carlos.cid}@rhul.ac.uk

1 Introduction

In [8] a method for extracting cryptographic key material from DRAM used in modern computers was proposed; the technique was called *Cold Boot attacks*. In the case of block ciphers, such as the AES and DES, simple algorithms were also proposed in [8] to recover the cryptographic key from the observed set of round subkeys in memory (computed via the cipher key schedule operation), which were however subject to errors (due to memory bits decay). In this work we propose methods for key recovery for other ciphers used in Full Disk Encryption (FDE) products. We demonstrate the viability of our method using the block-cipher SERPENT, which has a much more complex key schedule than AES. Our algorithms are also based on closest code word decoding methods, however apply a novel method for solving a set of non-linear algebraic equations with noise based on Integer Programming. This method should have further applications in cryptology, and is likely to be of independent interest.

2 Cold Boot Attacks

Cold Boot attacks were proposed in [8]. The method is based on the fact that DRAM may retain large part of its content for several seconds after removing its power, with gradual loss over that period. Furthermore, the time of retention can be potentially increased by reducing the temperature of memory. Thus contrary to common belief, data may persist in memory for several minutes after removal of power, subject to slow decay. As a result, data in DRAM can be used to recover potentially sensitive data, such as cryptographic keys, passwords, etc. A typical application is to defeat the security provided by disk encryption programs, such as Truecrypt [10]. In this case, cryptographic key material is maintained in memory, for transparent encryption and decryption of data. One could apply the method from [8] to obtain the computer's memory image, potentially extract the encryption key and then recover encrypted data.

* This author was supported by the Royal Holloway Valerie Myerscough Scholarship.

The Cold Boot attack has thus three stages: (a) the attacker physically removes the computer's memory, potentially applying cooling techniques to reduce the memory bits decay, to obtain the memory image; (b) locate cryptographic key material and other sensitive information in the memory image; and (c) recover the original cryptographic key. We refer the reader to [8, 9] for discussion on stages (a) and (b). In this work we concentrate on stage (c).

A few algorithms were proposed in [8] to tackle stage (c), which requires one to recover the original key based on the observed key material, probably subject to errors (the extent of which will depend on the properties of the memory, lapsed time from removal of power, and temperature of memory). In the case of block ciphers, the key material extracted from memory is very likely to be a set of round subkeys, which are the result of the cipher's key schedule operation. Thus the key schedule can be seen as an *error-correcting code*, and the problem of recovering the original key can be essentially described as a decoding problem.

The paper [8] contains methods for the AES and DES block ciphers. For DES, recovering the original 56-bit key is equivalent to decoding a repetition code. Textbook methods are used in [8] to recover the encryption key from the *closest code word* (i.e. valid key schedule). The AES key schedule is not as simple as DES, but still contains a large amount of linearity (which has also been exploited in recent related-key attacks, e.g. [3]). Another feature is that the original encryption key is used as the initial whitening subkey, and thus should be present in the key schedule. The method proposed in [8] for recovering the key for the AES-128 divides this initial subkey in four subsets of 32 bits, and uses 24 bits of the second subkey as redundancy. These small sets are then decoded in order of likelihood, combined and the resulting candidate keys are checked against the full schedule. The idea can be easily extended to the AES with 192- and 256-bit keys. The authors of [8] model the memory decay as a binary asymmetric channel, and recover an AES key up to error rates of $\delta_0 = 0.30, \delta_1 = 0.001$ (see notation below).

Other block ciphers are not considered in [8]. For instance, Serpent [2], formerly an AES candidate, is also found in popular FDE products (e.g. Truecrypt [10]). The cipher presents much more complex key schedule operations (with more non-linearity) than DES and AES. Another feature is that the original encryption key does not *explictly* appear in the expanded key schedule material (but rather has its bits non-linearly combined to derive the several round subkeys). These two facts led to the belief that these ciphers are not susceptible to the attacks in [8], and could be inherently more secure against Cold Boot attacks. In this work, we demonstrate that one can also recover the encryption key for the Serpent cipher. We expect that our methods can also be applied to other popular ciphers[1].

[1] In the full version of this paper we also consider the block cipher Twofish, for which we apply different, dedicated techniques.

3 The *Cold Boot* Problem

We define the *Cold Boot* problem as follows. Consider an efficiently computable vectorial Boolean function $\mathcal{KS} : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^N$ where $N > n$ and two real numbers $0 \leq \delta_0, \delta_1 \leq 1$. Let $K = \mathcal{KS}(k)$ be the image for some $k \in \mathbb{F}_2^n$, and K_i be the i-th bit of K. Now given K, compute $K' = (K'_0, K'_1, \ldots, K'_{N-1}) \in \mathbb{F}_2^N$ according to the following probability distribution:

$$\begin{array}{ll} Pr[K'_i = 0 \mid K_i = 0] = 1 - \delta_1 \ , & Pr[K'_i = 1 \mid K_i = 0] = \delta_1, \\ Pr[K'_i = 1 \mid K_i = 1] = 1 - \delta_0 \ , & Pr[K'_i = 0 \mid K_i = 1] = \delta_0. \end{array}$$

Thus we can consider such a K' as the output of $\mathcal{KS}$ for some $k \in \mathbb{F}_2^n$ except that K' is noisy. It follows that a bit $K'_i = 0$ of K' is correct with probability

$$Pr[K_i = 0 \mid K'_i = 0] = \frac{Pr[K'_i = 0 | K_i = 0] Pr[K_i = 0]}{Pr[K'_i = 0]} = \frac{(1 - \delta_1)}{(1 - \delta_1 + \delta_0)}.$$

Likewise, a bit $K'_i = 1$ of K' is correct with probability $\frac{(1-\delta_0)}{(1-\delta_0+\delta_1)}$. We denote these values by Δ_0 and Δ_1 respectively.

Now assume we are given the function $\mathcal{KS}$ and a vector $K' \in \mathbb{F}_2^N$ obtained by the process described above. Furthermore, we are also given an efficiently computable control function $\mathcal{E} : \mathbb{F}_2^n \rightarrow \{True, False\}$ which returns *True* or *False* for a given candidate k. The task is to recover k such that $\mathcal{E}(k)$ returns $True$. For example, $\mathcal{E}$ could use the encryption of some data to check whether the key k recovered is the original key.

In the context of this work, we can consider the function $\mathcal{KS}$ as the key schedule operation of a block cipher with n-bit keys. The vector K is the result of the key schedule expansion for a key k, and the noisy vector K' is obtained from K due to the process of memory bit decay. We note that in this case, another goal of the adversary could be recovering K rather than k (that is, the expanded key rather than the original encryption key), as with the round subkeys one could implement the encryption/decryption algorithm. In most cases, one should be able to efficiently recover the encryption key k from the expanded key K. However it could be conceivable that for a particular cipher with a highly non-linear key schedule, the problems are not equivalent.

Finally, we note that the *Cold Boot* problem is equivalent to decoding (potentially non-linear) binary codes with biased noise.

4 Solving Systems of Algebraic Equations with Noise

In this section we propose a method for solving systems of multivariate algebraic equations with noise. We use the method to implement a Cold Boot attack against ciphers with key schedule with a higher degree of non-linearity, such as Serpent.

4.1 Max-PoSSo

Polynomial system solving (*PoSSo*) is the problem of finding a solution to a system of polynomial equations over some field $\mathbb{F}$. We consider the set $F = \{f_0, \dots, f_{m-1}\}$, where each $f_i \in \mathbb{F}[x_0, \dots, x_{n-1}]$. A solution to F is any point $x \in \mathbb{F}^n$ such that $\forall f \in F$, we have $f(x) = 0$. Note that we restrict ourselves to solutions in the base field in the context of this work.

We can define a family of "Max-PoSSo" problems, analogous to the well-known Max-SAT family of problems. In fact, these problems can be reduced to their SAT equivalents. However, the modelling as polynomial systems seems more natural in our context. Thus let *Max-PoSSo* denote the problem of finding any $x \in \mathbb{F}^n$ that satisfies the maximum number of polynomials in F. Likewise, by *Partial Max-PoSSo* we denote the problem of returning a point $x \in \mathbb{F}^n$ such that for two sets of polynomials $\mathcal{H}$ and $\mathcal{S}$ in $\mathbb{F}[x_0, \dots, x_{n-1}]$, we have $f(x) = 0$ for all $f \in \mathcal{H}$, and the number of polynomials $f \in \mathcal{S}$ with $f(x) = 0$ is maximised. Max-PoSSo is Partial Max-PoSSo with $\mathcal{H} = \varnothing$.

Finally, by *Partial Weighted Max-PoSSo* we denote the problem of returning a point $x \in \mathbb{F}^n$ such that $\forall f \in \mathcal{H} : f(x) = 0$ and $\sum_{f \in \mathcal{S}} \mathcal{C}(f, x)$ is minimised, where $\mathcal{C} : f \in \mathcal{S}, x \in \mathbb{F}^n \to \mathbb{R}_{\geq 0}$ is a cost function which returns 0 if $f(x) = 0$ and some value $v > 0$ if $f(x) \neq 0$. Partial Max-PoSSo is Partial Weighted Max-PoSSo where $\mathcal{C}(f, x)$ returns 1 if $f(x) \neq 0$ for all f. We can consider the *Cold Boot* problem as a Partial Weighted Max-PoSSo problem over $\mathbb{F}_2$.

Let $F_{\mathcal{K}}$ be an equation system corresponding to $\mathcal{KS}$ such that the only pairs (k, K) that satisfy $F_{\mathcal{K}}$ are any $k \in \mathbb{F}_2^n$ and $K = \mathcal{K}(k)$. In our task however, we need to consider $F_{\mathcal{K}}$ with k and K'. Assume that for each noisy output bit K'_i there is some $f_i \in F_{\mathcal{K}}$ of the form $g_i + K'_i$, where g_i is some polynomial. Furthermore assume that these are the only polynomials involving the output bits ($F_{\mathcal{K}}$ can always be brought into this form) and denote the set of these polynomials $\mathcal{S}$. Denote the set of all remaining polynomials in $F_{\mathcal{K}}$ as $\mathcal{H}$, and define the cost function $\mathcal{C}$ as a function which returns

$$\begin{array}{cr} \frac{1}{1-\Delta_0} & \text{for } K'_i = 0, f(x) \neq 0, \\ \frac{1}{1-\Delta_1} & \text{for } K'_i = 1, f(x) \neq 0, \\ 0 & \text{otherwise.} \end{array}$$

Finally, let $F_{\mathcal{E}}$ be an equation system that is only satisfiable for $k \in \mathbb{F}_2^n$ for which $\mathcal{E}$ returns *True*. This will usually be an equation system for one or more encryptions. Add the polynomials in $F_{\mathcal{E}}$ to $\mathcal{H}$. Then $\mathcal{H}, \mathcal{S}, \mathcal{C}$ define a Partial Weighted Max-PoSSo problem. Any optimal solution x to this problem is a candidate solution for the *Cold Boot* problem.

In order to solve Max-PoSSo problems, we can reduce them to Max-SAT problems. However, in this work we consider a different approach which appears to better capture the algebraic structure of the underlying problems.

4.2 Mixed Integer Programming

Integer optimisation deals with the problem of minimising (or maximising) a function in several variables subject to linear equality and inequality constraints and integrality restrictions on some of or all the variables. A linear mixed integer programming problem (MIP) is defined as a problem of the form

$$\min_x \{c^T x | Ax \leq b, x \in \mathbb{Z}^k \times \mathbb{R}^l\},$$

where c is an n-vector ($n = k + l$), b is an m-vector and A is an $m \times n$-matrix. This means that we minimise the linear function $c^T x$ (the inner product of c and x) subject to linear equality and inequality constraints given by A and b. Additionally $k \geq 0$ variables are restricted to integer values while $l \geq 0$ variables are real-valued. The set S of all $x \in \mathbb{Z}^k \times \mathbb{R}^l$ that satisfy the linear constraints $Ax \leq b$, that is

$$S = \{x \in \mathbb{Z}_k \times \mathbb{R}_l \mid Ax \leq b\},$$

is called the feasible set. If $S = \varnothing$ the problem is infeasible. Any $x \in S$ that minimises $c^T x$ is an optimal solution.

We can convert the PoSSo problem over $\mathbb{F}_2$ to a mixed integer programming problem using the Integer Adapted Standard Conversion described in [4].

Furthermore we can convert a Partial Weighted Max-PoSSo problem into a Mixed Integer Programming problem as follows. Convert each $f \in \mathcal{H}$ to linear constraints as in [4]. For each $f_i \in \mathcal{S}$ add some new binary slack variable e_i to f_i and convert the resulting polynomial as before. The objective function we minimise is $\sum c_i e_i$, where c_i is the value of $\mathcal{C}(f, x)$ for some x such that $f(x) \neq 0$. Any optimal solution $x \in S$ will be an optimal solution to the Partial Weighted Max-PoSSo problem.

We note that this approach is essentially the non-linear generalisation of decoding random linear codes with linear programming [6].

5 Cold Boot Key Recovery

We have applied the method discussed above to implement a *Cold Boot* key recovery attack against AES and SERPENT. We refer the reader to [5, 2] for the details of the corresponding key schedule operations. We will focus on the 128-bit versions of the two ciphers.

For each instance of the problem, we performed our experiments with between $40 - 100$ randomly generated keys. In the experiments we usually did not consider the full key schedule but rather a reduced number of rounds in order to improve the running time of our algorithms. We also did not include equations for $\mathcal{E}$ explicitly. Finally, we also considered at times an "aggressive" modelling, where we set $\delta_1 = 0$ instead of $\delta_1 = 0.001$. In this case all values $K'_i = 1$ are considered correct (since $\Delta_1 = 1$), and as a result all corresponding equations are promoted to the set $\mathcal{H}$.

5.1 Experimental Results

In the Appendix we give running times and success rates for key recovery using the SERPENT key schedule up to $\delta_0 = 0.30$. We also give running times and success rates for the AES up to $\delta_0 = 0.40$ (in order to compare our approach with that in [8], where error rates up to $\delta_0 = 0.30$ were considered). We note that in the case of the AES, a success rate lower than 100% may still allow a successful key recovery since the algorithm can be run on later segments of the key schedule if it fails for the first few rounds.

Acknowledgements

We would like to thank Stefan Heinz, Timo Berthold and Ambros Gleixner from the SCIP team for optimising our parameters and helpful discussions. We would also like to thank an anonymous referee for helpful comments.

References

1. Tobias Achterberg. Constraint Integer Programming. PhD thesis, TU Berlin 2007. `http://scip.zib.de`
2. E. Biham, R.J. Anderson, and L.R. Knudsen. SERPENT: A New Block Cipher Proposal. In S. Vaudenay, editor, *Fast Software Encryption 1998*, volume 1372 of *LNCS*, pages 222–238. Springer–Verlag, 1998.
3. Alex Biryukov, Dmitry Khovratovich and Ivica Nikolić. Distinguisher and Related-Key Attack on the Full AES-256. *Advances in Cryptology - CRYPTO 2009*, LNCS 5677, 231–249, Springer Verlag 2009.
4. Julia Borghoff, Lars R. Knudsen and Mathias Stolpe. Bivium as a Mixed-Integer Linear Programming Problem. *Cryptography and Coding – 12th IMA International Conference*, LNCS 5921, 133–152, Springer Verlag 2009.
5. J. Daemen and V. Rijmen. *The Design of Rijndael.* Springer–Verlag, 2002.
6. Jon Feldman. Decoding Error-Correcting Codes via Linear Programming. PhD thesis, Massachusetts Institute of Technology 2003.
7. Gurobi Optimization, Inc, `http://gurobi.com`.
8. J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino and Ariel J. Feldman, Jacob Appelbaum and Edward W. Felten. Lest We Remember: Cold Boot Attacks on Encryption Keys. *USENIX Security Symposium*, 45–60, USENIX Association 2009.
9. Nadia Heninger and Hovav Shacham. Reconstructing RSA Private Keys from Random Key Bits. *Cryptology ePrint Archive, Report 2008/510*, 2008.
10. TrueCrypt Project, `http://www.truecrypt.org/`.

A Experimental Results

Running times for the MIP solvers Gurobi [7] and SCIP [1] are given below. For SCIP the tuning parameters were adapted to meet our problem[2]; no such optimisation was performed for Gurobi. Column "a" denotes either aggressive ("+") or normal ("–") modelling. The column "cutoff t" denotes the time we maximally allowed the solver to run until we interrupted it. The column r gives the success rate.

		Gurobi				
N	δ_0	a	#cores	cutoff t	r	max t
3	0.15	–	24	∞	100%	17956.4s
3	0.15	–	2	240.0s	25%	240.0s
3	0.30	+	24	3600.0s	25%	3600.0s
3	0.35	+	24	28800.0s	30%	28800.0s
		SCIP				
3	0.15	+	1	3600.0s	65%	1209.0s
4	0.30	+	1	7200.0s	47%	7200.0s
4	0.35	+	1	10800.0s	45%	10800.0s
4	0.40	+	1	14400.0s	52%	14400.0s
5	0.40	+	1	14400.0s	45%	14400.0s

Table 1. AES considering N rounds of key schedule output

		Gurobi				
N	δ_0	a	#cores	cutoff t	r	Max t
8	0.05	–	2	60.0s	50%	16.22s
12	0.05	–	2	60.0s	85%	60.00s
8	0.15	–	24	600.0s	20%	103.17s
12	0.15	–	24	600.0s	55%	600.00s
12	0.30	+	24	7200.0s	20%	7200.00s
		SCIP				
12	0.15	+	1	600.0s	32%	597.37s
16	0.15	+	1	3600.0s	48%	369.55s
20	0.15	+	1	3600.0s	29%	689.18s
16	0.30	+	1	3600.0s	55%	3600.00s
20	0.30	+	1	7200.0s	57%	7200.00s

Table 2. SERPENT considering $32 \cdot N$ bits of key schedule output

[2] `branching/relpscost/maxreliable = 1`, `branching/relpscost/inititer = 1`, `separating/cmir/maxroundsroot = 0`

Practical Key Recovery Attacks On Two McEliece Variants

Valérie Gauthier Umaña and Gregor Leander

Department of Mathematics
Technical University of Denmark
Denmark
{v.g.umana, g.leander}@dtu.mat.dk

Abstract. The McEliece cryptosystem is a promising alternative to conventional public key encryption systems like RSA and ECC. In particular, it is supposed to resist even attackers equipped with quantum computers. Moreover, the encryption process requires only simple binary operations making it a good candidate for low cost devices like RFID tags. However, McEliece's original scheme has the drawback that the keys are very large. Two promising variants have been proposed to overcome this disadvantage. The first one is due to Berger et al. presented at AFRICACRYPT 2009 and the second is due to Barreto and Misoczki presented at SAC 2009. In this paper we first present a general attack framework and apply it to both schemes subsequently. Our framework allows us to recover the private key for most parameters proposed by the authors of both schemes within at most a few days on a single PC.[1]

Keywords. public key cryptography, McEliece cryptosystem, coding theory, post-quantum cryptography

1 Introduction

Today, many strong, and standardized, public key encryption schemes are available. The most popular ones are based on either the hardness of factoring or of computing discrete logarithms in various groups. These systems provide an excellent and preferable choice in many applications, their security is well understood and efficient (and side-channel resistant) implementations are available.

However, there is still a strong need for alternative systems. There are at least two important reasons for this. First, RSA and most of the discrete log based cryptosystems would break down as soon as quantum computers of an appropriate size could be built (see [11]). Thus, it is desirable to have algorithms at hand that would (supposedly) resist even attackers equipped with quantum computers. The second reason is that most of the standard schemes are too costly to be implemented in very constrained devices like RFID tags or sensor networks. This issue becomes even more important when looking at the future IT landscape where it is anticipated that those tiny computer devices will be virtually everywhere [13].

One well-known alternative public encryption scheme that, to today's knowledge, would resist quantum computers is the McEliece crypto scheme [9]. It is based on the hardness of decoding random (looking) linear codes. Another advantage is that for encryption only elementary binary operations are needed and one might therefore hope that McEliece is suitable for constrained devices (see also [5] for recent results in this direction). However, the original McEliece scheme has a serious drawback, namely the public and the secret keys are orders of magnitude larger compared to RSA and ECC.

One very reasonable approach is therefore to modify McEliece's original scheme in such a way that it remains secure while the key size is reduced. A lot of papers already followed that line of

[1] Note that, in [6] Faugère et.al. present independent analysis of the same two schemes with better running times compared to our attacks.

research (see for example [10,7,1]), but so far no satisfying answer has been found. Two promising recent schemes in this direction are a McEliece variant based on quasi-cyclic alternant codes by Berger et al. [3] and a variant based on quasi-dyadic matrices by Barreto and Misoczki [2]. As we will explain below both papers follow a very similar approach to find suitable codes. The reduction in the key size of both schemes is impressive and thus, those schemes seemed to be very promising alternatives when compared to RSA and ECC.

In this paper, however, we show that many of the parameter choices for both schemes can be broken. For some of them the secret key can be computed, given the public one, within less than a minute for the first variant and within a few hours for the second scheme. While there remain some parameter choices that we cannot attack efficiently, it seems that further analysis is needed before those schemes should be used.

Our attack is based on a general framework that makes use of (linear) redundancy in subfield subcodes of generalized Reed Solomon Codes. We anticipate therefore that any variant that reduces the key size by introducing redundancy in such (or related) codes might be affected by our attack as well.

We describe the two variants in Section 3 briefly, giving only the details relevant for our attack (for more details we refer the reader to the papers [3,2]). In Section 4 we outline the general framework of the attack and finally apply this framework to both schemes (cf. Section 5 and 6).

2 Notation

Let r, m be integers and $q = 2^r$. We denote by $\mathbb{F}_q$ the finite field with q elements and by $\mathbb{F}_{q^m}$ its extension of degree m. In most of the cases we will consider the case $m = 2$ and we stick to this until otherwise stated. For an element $x \in \mathbb{F}_{q^2}$ we denote its conjugate x^q by $\overline{x}$.

Given an $\mathbb{F}_q$ basis $1, \omega$ of $\mathbb{F}_{q^2}$ we denote by $\psi : \mathbb{F}_{q^2} \to \mathbb{F}_q^2$ the vector space isomorphism such that $\psi(x) = \psi(x_0 + \omega x_1) = \binom{x_1}{x_0}$. Note that, without loss of generality, we can choose θ such that $\psi(x) = \binom{\phi(x)}{\phi(\theta x)}$ where $\phi(x) = x + \overline{x}$ with $\overline{x} = x^q$. Note that we have the identity

$$\phi(\overline{x}) = \phi(x). \tag{1}$$

A fact that we will use at several instances later is that given $a = \phi(\alpha x)$ and $b = \phi(\beta x)$ for some $\alpha, \beta, x \in \mathbb{F}_{q^2}$ we can recover x as linear combination of a and b (as long as α, β form an $\mathbb{F}_q$ basis of $\mathbb{F}_{q^2}$). More precisely it holds that

$$x = \frac{\overline{\alpha}}{\beta\overline{\alpha} + \overline{\beta}\alpha} b + \frac{\overline{\beta}}{\beta\overline{\alpha} + \overline{\beta}\alpha} a \tag{2}$$

Adopting the notation from coding theory, all vectors in the papers are row vectors and vectors are right multiplied by matrices. The i.th component of a vector x is denote by $x^{(i)}$. Due to space limitations, we do not recall the basis concepts of coding theory on which McEliece and the two variants are based on. They are not needed for our attack anyway. Instead we refer the reader to [8] for more background on coding theory and in particular on subfield subcodes of Generalized Reed Solomon codes.

3 Two McEliece Variants

We are going to introduce briefly the two McEliece variants [3,2] that we analyzed. For this, we denote by x_i, c_i two sets of elements in $\mathbb{F}_{q^2}$ of size n. Furthermore let t be an integer. Both variants have a secret key parity check matrix of the form:

$$\textbf{(secret key)} \quad H = \begin{pmatrix} \phi(c_0) & \phi(c_1) & \dots & \phi(c_{n-1}) \\ \phi(\theta c_0) & \phi(\theta c_1) & \dots & \phi(\theta c_{n-1}) \\ \vdots & \vdots & & \vdots \\ \phi(c_0 x_0^{t-1}) & \phi(c_1 x_1^{t-1}) & \dots & \phi(c_{n-1} x_{n-1}^{t-1}) \\ \phi(\theta c_0 x_0^{t-1}) & \phi(\theta c_1 x_1^{t-1}) & \dots & \phi(\theta c_{n-1} x_{n-1}^{t-1}) \end{pmatrix} = \begin{pmatrix} \mathrm{sk}_0 \\ \vdots \\ \mathrm{sk}_{2t-1} \end{pmatrix} \tag{3}$$

Thus, both schemes are based on subfield subcodes of Generalized Reed Solomon codes (see [8] for more background on those codes). Note that Goppa codes, the basic building block for the original McEliece encryption scheme are a particular kind of subfield subcodes of Generalized Reed Solomon codes. To simplify the notation later we denote by sk_i the i.th row of H.

The public key in both variants is

$$\textbf{(public key)} \quad P = SH, \tag{4}$$

where S is a secret invertible $2t \times 2t$ matrix. Actually, in both schemes P is defined to be the systematic form of H, which leads to a special choice of S. As we do not make use of this fact for the attacks one might as well consider S as a random invertible matrix.

In both cases, without loss of generality c_0 and x_0 can be supposed to be 1. In fact, given that the public key H is not uniquely defined, we can always include the corresponding divisions needed for this normalization into the matrix S.

The main difference between the two proposals is the choice of the constants c_i and the points x_i. In order to reduce the keysize, both of the public as well as of the secret key, those $2n$ values are not chosen independently, but in a highly structured way.

Both schemes use random block-shortening of very large private codes (exploiting the NP-completeness of distinguishing punctured codes [14]) and the subfield subcode construction (to resist the classical attack of Sidelnikov and Shestakov, see [12]). In [3,2] the authors analyze the security of their schemes and demonstrate that none of the known attack can be applied. They also prove that the decoding of an arbitrary quasi-cyclic (reps. an arbitrary quasi-dyadic) code is NP-complete.

For the subfield subcode construction, both schemes allow in principle any subfield to be used. However the most interesting case in terms of key size and performance is the case when the subfield is of index 2 (i.e. $m = 2$) and we focus on this case only.

Both schemes use a block based description of the secret codes. They take b blocks of ℓ columns and t rows. The subfield subcode operation will transform each block into a $2t \times \ell$ matrix and the secret parity check matrix H is the concatenation of the b blocks. Thus, one obtains a code of length ℓb.

Note that when we describe the variants, our notation will differ from the one in [3,2]. This is an inconvenience necessary in order to unify the description of our attack on both variants.

3.1 The Quasi-Cyclic Variant

Berger et al. propose [3] to use quasi-cyclic alternant codes over a small non-binary field. Let α be a primitive element of $\mathbb{F}_{q^m}$ and $\beta \in \mathbb{F}_{q^m}$ an element of order ℓ (those are public values). The secret key consists of b different values y_j and a_j in $\mathbb{F}_{q^m}$ where b is small, i.e. $b \leq 15$ for the proposed parameters. The constants c_i and points x_i are then defined by

$$c_{\ell j+i} := \beta^{is} a_j \quad \text{and} \quad x_{\ell j+i} := \beta^i y_j \tag{5}$$

for all $0 \leq i \leq \ell - 1$ and $0 \leq j \leq b-1$. Here $1 \leq s \leq \ell - 1$ is a secret value. Table 1 lists the parameters proposed in [3]. Note that in [3] cyclic shifts (modulo ℓ) of the columns are applied. This does not change the structure of the matrix (since β has order ℓ) and that is why we can omit this from our analysis.

Table 1. Parameters proposed in [3] and the running time/complexity of our attacks. The attacks were carried on a PC with an Intel Core2 Duo with 2.2 GHz and 3 GB memory running MAGMA version V2.15 − 12. Times are averaged over 100 runs.

	q	q^m	ℓ	t	b	Public key size (bits)	Assumed security	Complexity ($\log_2$) attack Section 5.1	Av. running time (sec) attack Section 5.2	Av. running time (sec) attack Appendix B
I			51	100	9	8160	80	74.9	–	–
II			51	100	10	9792	90	75.1	–	–
III	2^8	2^{16}	51	100	12	13056	100	75.3	–	–
IV			51	100	15	20400	120	75.6	–	–
V			75	112	6	6750	80	–	–	47
VI	2^{10}	2^{20}	93	126	6	8370	90	87.3	62	–
VII			93	108	8	14880	100	86.0	75	–

3.2 The Quasi-Dyadic Variant

Misoczki and Barreto propose [2] to use binary Goppa codes in dyadic form. They consider (quasi) dyadic Cauchy matrices as the parity check matrix for their code. However, it is well known that Cauchy matrices define generalized Reed Solomon codes in field of characteristic 2 [2] and thus, up to a multiplication by an invertible matrix which we consider to be incorporated in the secret matrix S, the scheme has a parity check matrix of the form (3).

Again, the only detail to be described here is how the constants c_i and points x_i are chosen. First we choose $\ell = t$ a power of two. Next, choose $v = [\mathbb{F}_{q^m} : \mathbb{F}_2] = mr$ elements in $\mathbb{F}_{q^m}$: $y_0, y_1, y_2, y_4, \cdots, y_{2^v}$. For each $j = \sum_{k=0}^{v} j_k 2^k$ such that $j_k \in \{0, 1\}$ (i.e. the binary representation of j) we define

$$y_j = \sum_{k=0}^{v} j_k y_{2^k} + (W_H(j) + 1) y_0 \tag{6}$$

for $0 \leq j \leq \#\mathbb{F}_{q^m} - 1$ and $W_H(j)$ is the Hamming weight of j. Moreover, choose b different elements k_i with $0 \leq i \leq \#\mathbb{F}_{q^m} - 1$, b different elements $a_i \in \mathbb{F}_{q^m}$ and define

$$x_{\ell i+j} := y_{k_i \oplus j} \quad \text{and} \quad c_{\ell i+j} := a_i \tag{7}$$

for all $0 \leq j \leq \ell - 1$ and $0 \leq i \leq b-1$. This choice implies the following identity. For $j = \sum_{f=0}^{u-1} j_f 2^f$, where $u = \log_2(\ell)$ it holds that

$$x_{\ell i+j} = \sum_{f=0}^{u-1} j_f x_{\ell i+2^f} + (W_H(j)+1)x_{li}. \tag{8}$$

Note that in [2] dyadic permutations are applied. However, this does not change the structure of the matrix and that is why we can omit this from our analysis.

Table 2 lists the parameters proposed in [2, Table 5].

Table 2. Sample parameters from [2] along with the complexity of our attack. Running time was measured on a PC with an Intel Core2 Duo with 2.2 GHz and 3 GB memory running MAGMA version V2.15 − 12.

q	q^m	ℓ	t	b	public key size	assumed security	complexity of the attack ($\log_2$)	estimated running time(h)
2^8	2^{16}	128	128	4	4096	80	43.7	36
		128	128	5	6144	112	43.8	41
		128	128	6	8192	128	44.0	47
		256	256	5	12288	192	44.8	107
		256	256	6	16384	256	44.9	125

4 General Framework of the Attack

The starting observation for our analysis and attacks is the following interpretation of the entries in the public key P.

Proposition 1. *Let H be the $2t \times n$ parity check matrix defined as in Equation (3). Multiplying H by a $2t \times 2t$ matrix S we obtain a $2t \times n$ matrix P of the form*

$$P = SH = \begin{pmatrix} \phi(c_0 g_0(x_0)) & \phi(c_1 g_0(x_1)) & \dots & \phi(c_{n-1} g_0(x_{n-1})) \\ \phi(c_0 g_1(x_0)) & \phi(c_1 g_1(x_1)) & \dots & \phi(c_{n-1} g_1(x_{n-1})) \\ \vdots & \vdots & & \vdots \\ \phi(c_0 g_{2t-1}(x_0)) & \phi(c_1 g_{2t-1}(x_1)) & \dots & \phi(c_{n-1} g_{2t-1}(x_{n-1})) \end{pmatrix}$$

where g_i are polynomials with coefficients in $\mathbb{F}_{q^2}$ of degree less than t. Moreover, if S is bijective the polynomials g_i form an $\mathbb{F}_q$ basis of all polynomials of degree at most $t-1$.

The proof of this proposition can be found in Appendix A.

This observation allows us to carry some of the spirit of the attack of Sidelnikov and Shestakov (see [12]) on McEliece variants based on Reed-Solomon codes over to the subfield subcode case. The basic idea is that multiplying the public key P by a vector results (roughly speaking) in the evaluation of a polynomial at the secret points x_i. More precisely the following holds.

Proposition 2. *Continuing the notation from above, multiplying the public parity check matrix P with a vector $\gamma \in \mathbb{F}_q^{2t}$ results in*

$$\gamma P = (\phi(c_0 g_\gamma(x_0)), \dots, \phi(c_{n-1} g_\gamma(x_{n-1}))) \tag{9}$$

where $g_\gamma(x) = \sum_{i=0}^{2t-1} \gamma_i g_i(x)$.

Table 3. The relation among the values γ, g_γ and γP. The polynomials g_i are defined in Proposition 1

γ	A vector in $\mathbb{F}_q^{2t}$
g_γ	The polynomial defined by $g_\gamma(x) = \sum_{i=0}^{2t-1} \gamma_i g_i(x)$.
γP	A vector in $\mathbb{F}_q^n$ whose entries are given by $\phi(c_i g_\gamma(x_i))$.

As the values γ, g_γ and γP are extensively used below we summarize their relation in Table 3.

Thus, if we would have the possibility to control the polynomial g_γ (even though we do not know the polynomials g_i) then γP reveals, hopefully, useful information on the secret key. While in general, controlling g_γ seems difficult, it becomes feasible in the case where the secret points x_i and the constants c_i are not chosen independently, but rather fulfil (linear) relations. The attack procedure can be split into three phases.

Isolate: The first step of the attack consists in choosing polynomials g_γ that we want to use in the attack. The main obstacle here is that we have to choose g_γ such that the redundancy allows us to efficiently recover the corresponding γ. As we will see later, it is usually not possible to isolate a single polynomial g_γ but rather to isolate a vector space of polynomials (or, equivalently, of vectors γ) of sufficiently small dimension.

Collect: After the choice of a set of polynomials and the recovery of the corresponding vectors γ, the next step of the attack consists in evaluating those polynomials at the secret points x_i. In the light of Proposition 2 this is simply done by multiplying the vectors γ with the public parity check matrix P.

Solve: Given the information collected in the second step of the attack, we now have to extract the secret key, i.e. the values x_i and c_i. This corresponds to solving a system of equations. Depending on the type of collected information this is done simply by solving linear equations, by first guessing parts of the key and then verifying by solving linear equations, or by solving non-linear equations with the help of Gröbner basis techniques. The advantage of the first two possibilities is that one can easily determine the running time in general while this is not true for the last one. However, the use of Gröbner basis techniques allows us to attack specific parameters very efficiently.

4.1 The Isolate Phase and the Collect Phase in Detail

The redundancy in the choice of the points x_i and the constants c_i will allow us to identify sets of polynomials or more precisely vector spaces of polynomials. In this section we elaborate a bit more on this on a general level. Assume that we are able to identify a subspace $\Gamma \subseteq \mathbb{F}_q^{2t}$ such that for each $\gamma \in \Gamma$ we know that g_γ is of the form

$$g_\gamma = \alpha_1 x^{d_1} + \alpha_2 x^{d_2} + \cdots + \alpha_r x^{d_r}$$

for some $\alpha_i \in \mathbb{F}_{q^2}$ and $d_i < t$. Equation (9) states that multiplying γ with the public key yields

$$\gamma P = (\phi(c_0 g_\gamma(x_0)), \ldots, \phi(c_{n-1} g_\gamma(x_{n-1})))\,.$$

Using the assumed form of g_γ, and writing $\alpha_i = \alpha_{i,1} + \alpha_{i,2}\theta$ with $\alpha_{i,1}, \alpha_{i,2} \in \mathbb{F}_q$, we can rewrite $\phi(c g_\gamma(x))$ as

$$\begin{aligned}\phi(c g_\gamma(x)) &= \phi(c(\alpha_1 x^{d_1} + \alpha_1 x^{d_1} + \ldots \alpha_r x^{d_r})) \\ &= \alpha_{1,1}\phi(c x^{d_1}) + \alpha_{1,2}\phi(\theta c x^{d_1}) + \cdots + \alpha_{r,1}\phi(c x^{d_r}) + \alpha_{r,2}\phi(\theta c x^{d_r}).\end{aligned}$$

Recalling that we denote by sk_i the i.th row of the secret key (cf. Equation 3), we conclude that

$$\gamma P = \alpha_{1,1}\,\mathrm{sk}_{2d_1} + \alpha_{1,2}\,\mathrm{sk}_{2d_1+1} + \alpha_{2,1}\,\mathrm{sk}_{2d_2} + \alpha_{2,2}\,\mathrm{sk}_{2d_2+1} + \cdots + \alpha_{r,2}\,\mathrm{sk}_{2d_r+1}\,.$$

Now, if the dimension of Γ is $2r$ this implies that there is a one to one correspondence between the elements $\gamma \in \Gamma$ and the coefficient vector $(\alpha_1, \ldots, \alpha_r)$. Stated differently, there exists an invertible $2r \times 2r$ matrix M such that for a basis $\gamma_1, \ldots, \gamma_{2r}$ of Γ we have

$$\begin{pmatrix} \gamma_1 \\ \vdots \\ \gamma_{2r} \end{pmatrix} P = M \begin{pmatrix} \mathrm{sk}_{2d_1} \\ \vdots \\ \mathrm{sk}_{2d_r+1} \end{pmatrix}, \tag{10}$$

where we now know all the values on the left side of the equation. This has to be compared to the initial problem (cf Equation 4) we are facing when trying to recover the secret key given the public one, where S is an invertible $2t \times 2t$ matrix. In this sense, the first step of the attack allows us to break the initial problem into (eventually much) smaller subproblems. Depending on the size of r (which will vary between 1 and $\log_2 t$ in the actual attacks) and the specific exponents d_i involved, this approach will allow us to efficiently reconstruct the secret key.

Note that we are actually not really interested in the matrix M, but rather in the values x_i and c_i. Therefore, a description of the result of the isolate and collect phase that is often more useful for actually solving for those unknowns is given by

$$M^{-1} \begin{pmatrix} \gamma_1 \\ \vdots \\ \gamma_{2r} \end{pmatrix} P = \begin{pmatrix} \mathrm{sk}_{2d_1} \\ \vdots \\ \mathrm{sk}_{2d_r+1} \end{pmatrix}. \tag{11}$$

The advantage of this description is that the equations are considerably simpler (in particular linear in the entries of M^{-1}) as we will see when attacking specific parameters.

5 Applying the Framework to the Quasi-Cyclic Variant

In the following we show how the framework described above applies to the McEliece variant from [3] defined in Section 3.1. In particular we are going to make use of Equation (5). Recall that β is an element of order ℓ in $\mathbb{F}_{q^2}$. If ℓ is a divisor of $q-1$, such an element is in the subfield $\mathbb{F}_q$. This is the case for all the parameters in Table 1 except the parameter set V. We first focus on this case, the case that β is not in the subfield is considered in Appendix B. Section 5.1 describes an attack that works for parameters I-IV,VI and VII. Furthermore, for parameters VI and VII we describe attacks that allow us to recover the secret key within a few seconds in Section 5.2.

Note that in any case the secret value s (cf. Equation (5)) can be recovered very efficiently before applying the actual attacks, and we therefore assume it to be known from now on. However, due to space limitations and the fact that s is small anyway, we do not explain the details for recovering s.

5.1 The case $\beta \in \mathbb{F}_q$ (parameters I-IV,VI and VII)

In this part we describe an attack that works essentially whenever β is in the subfield. The attack has a complexity of roughly $q^6 \times (n_d b)(4n_d + b)^2(\log_2 q^2)^3$ (where $n_d = \lfloor \log_2(t-\ell) \rfloor$) which is lower than the best attacks known so far. Moreover, the attack is a key recovery attack, thus running the attack once allows an attacker to efficiently decrypt any ciphertext. However, these attacks are far from being practical (cf. Table 1 for actual values).

In the attack we apply the general framework twice. The first part will reduce the number of possible constants c_i to q^6 values. In the second part, for each of those possibilities, we try to find the points x_i by solving an over defined system of linear equations. This system will be solvable for the correct constants and in this case reveal the secret points x_i.

Recovering the Constants c_j

Isolate: We start by considering the simplest possible candidate for g_γ, namely $g_\gamma(x) = 1$. The task now is to compute the corresponding vector γ. Multiplying the desired vector γ with the public key P we expect (cf. Equation (9)) to obtain the following

$$\gamma P = (\phi(c_0 g_\gamma(x_0)), \dots, \phi(c_{n-1} g_\gamma(x_{n-1}))) = (\phi(c_0), \phi(c_1), \dots, \phi(c_{n-1})).$$

Now, taking Equation (5) into account, this becomes

$$\begin{aligned}\gamma P = \Big(&\phi(a_0), \phi(\beta^s a_0), \phi(\beta^{2s} a_0), \dots, \phi(\beta^{(\ell-1)s} a_0),\\ &\phi(a_1), \phi(\beta^s a_1), \phi(\beta^{2s} a_1), \dots, \phi(\beta^{(\ell-1)s} a_1),\\ &\ \vdots \quad \vdots\\ &\phi(a_{b-1}), \phi(\beta^s a_{b-1}), \phi(\beta^{2s} a_{b-1}), \dots, \phi(\beta^{(\ell-1)s} a_{b-1})\Big).\end{aligned}$$

Since β is in the subfield we have $\phi(\beta x) = \beta \phi(x)$ for any $x \in \mathbb{F}_{q^2}$. Using this identity we see that γ corresponding to the constant polynomial g_γ fulfils

$$\gamma P = \phi(a_0) v_0 + \phi(a_1) v_1 + \cdots + \phi(a_{b-1}) v_{b-1}$$

where

$$v_i = (\underbrace{0, \dots, 0}_{i\ell}, 1, \beta^s, \beta^{2s}, \dots, \beta^{(\ell-1)s}, \underbrace{0, \dots 0}_{((b-1)-i)\ell}) \quad \text{for } 0 \le i \le b-1.$$

In other words, the γ we are looking for is such that γP is contained in the space U spanned by v_0 up to v_{b-1}, i.e. $\gamma P \in U = \langle v_0, \dots, v_{b-1} \rangle$. Thus to compute candidates for γ we have to compute a basis for the space $\Gamma_0 = \{\gamma \mid \gamma P \in U\}$. We computed this space for many randomly generated public keys and observed the following.

Experimental Observation 1 *The dimension of the space Γ_0 is always* 4.

We do not prove this, but the next lemma explains why the dimension is at least 4.

Lemma 1. *Let γ be a vector such that $g_\gamma(x) = \alpha_0 + \alpha_1 x^\ell$. Then $\gamma \in \Gamma_0$.*

Proof. To show that γ is in Γ_0 we have to show that γP is a linear combination of the vectors v_i. To see this, it suffices to note that $g_\gamma(\beta x) = \alpha_0 + \alpha_1 (\beta x)^\ell = \alpha_0 + \alpha_1 x^\ell = g_\gamma(x)$ as $\beta^\ell = 1$. As the points x_i fulfil Equation (5) we conclude $\gamma P = \phi(a_0 g_\gamma(y_0)) v_0 + \phi(a_1 g_\gamma(y_1)) v_1 + \cdots + \phi(a_{b-1} g_\gamma(y_{b-1})) v_{b-1}$. □

As, due to Observation 1, $\dim(\Gamma_0) = 4$ we conclude that

$$\{g_\gamma \mid \gamma \in \Gamma_0\} = \{\alpha_0 + \alpha_1 x^\ell \mid \alpha_0, \alpha_1 \in \mathbb{F}_{q^2}\}.$$

Collect Phase: Denote by $\gamma_1, \ldots, \gamma_4$ a basis of the four dimensional space Γ_0. Referring to Equation (11) we get

$$M^{-1} \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \end{pmatrix} P = \begin{pmatrix} \mathrm{sk}_0 \\ \mathrm{sk}_1 \\ \mathrm{sk}_{2\ell} \\ \mathrm{sk}_{2\ell+1} \end{pmatrix}. \tag{12}$$

for an (unknown) 4×4 matrix M^{-1} with coefficients in $\mathbb{F}_q$.

Solve Phase: We denote the entries of M^{-1} by (β_{ij}). The i.th component of the first two rows of Equation (12) can be rewritten as

$$\beta_{00}(\gamma_1 P)^{(i)} + \beta_{01}(\gamma_2 P)^{(i)} + \beta_{02}(\gamma_3 P)^{(i)} + \beta_{03}(\gamma_4 P)^{(i)} = \mathrm{sk}_0^{(i)} = \phi(c_i) = c_i + \overline{c_i}$$
$$\beta_{10}(\gamma_1 P)^{(i)} + \beta_{11}(\gamma_2 P)^{(i)} + \beta_{12}(\gamma_3 P)^{(i)} + \beta_{13}(\gamma_4 P)^{(i)} = \mathrm{sk}_1^{(i)} = \phi(\theta c_i) = \theta c_i + \overline{\theta c_i}.$$

Dividing the second equation by $\overline{\theta}$ and adding the two implies

$$\delta_0(\gamma_1 P)^{(i)} + \delta_1(\gamma_2 P)^{(i)} + \delta_2(\gamma_3 P)^{(i)} + \delta_3(\gamma_4 P)^{(i)} = \left(\frac{\theta}{\overline{\theta}} + 1\right) c_i, \tag{13}$$

where

$$\delta_i = \left(\beta_{0i} + \frac{\beta_{1i}}{\overline{\theta}}\right) \in \mathbb{F}_{q^2}.$$

Assume without loss of generality that $c_0 = 1$. Then, for each possible choice of δ_0, δ_1 and δ_2 we can compute δ_3 (using $c_0 = 1$) and subsequently candidates for all constants c_i. We conclude that there are $(q^2)^3$ possible choices for the constants c_i (and thus in particular for the b constants $a_0 = c_0, \ldots, a_{b-1} = c_{(b-1)\ell}$). We will have to repeat the following step for each of those choices.

Recovering Points $\boldsymbol{x_i}$ Given one out of the q^6 possible guesses for the constants c_i we now explain how to recover the secret values x_i by solving an (over defined) system of linear equations. Most of the procedure is very similar to what was done to (partially) recover the constants.

Isolate Here we make use of polynomials $g_\gamma = x^d$ for $d \leq t-1$. The case $g_\gamma = 1$ is thus a special case $d = 0$. Following the same computations as above, we see that for the vector γ corresponding to $g_\gamma = 1$ it holds that $\gamma P \in U_d$ where

$$U_d = \langle v_{(d)0}, \ldots, v_{(d)b-1} \rangle \tag{14}$$

and

$$v_{(d)i} = (\underbrace{0, \ldots, 0}_{i\ell}, 1, \beta^{s+d}, \beta^{2(s+d)}, \ldots, \beta^{(\ell-1)(s+d)}, \underbrace{0, \ldots 0}_{((b-1)-i)\ell}) \quad \text{for } 0 \leq i \leq b-1.$$

As before we define $\Gamma_d = \{\gamma \mid \gamma P \in U_d\}$, and, based on many randomly generated public keys we state the following.

Experimental Observation 2 *For $d \leq t - \ell$ the dimension of the space Γ_d is always* 4.

Similar as above, the next lemma, which can be proven similar as Lemma 1, explains why the dimension of Γ_d is at least 4.

Lemma 2. *Let γ be a vector such that $g_\gamma(x) = \alpha_0 x^d + \alpha_1 x^{d+\ell}$. Then $\gamma \in \Gamma_d$.*

As, due to Observation 2, $\dim(\Gamma_d) = 4$ we conclude that

$$\{g_\gamma \mid \gamma \in \Gamma_d\} = \{\alpha_0 x^d + \alpha_1 x^{d+\ell} \mid \alpha_0, \alpha_1 \in \mathbb{F}_{q^2}\}.$$

Collect Phase: Denote by $\gamma_{(d)1}, \dots \gamma_{(d)4}$ a basis of the four dimensional space Γ_d. Referring to Equation (11) we get

$$M_d^{-1} \begin{pmatrix} \gamma_{(d)1} \\ \gamma_{(d)2} \\ \gamma_{(d)3} \\ \gamma_{(d)4} \end{pmatrix} P = \begin{pmatrix} \mathrm{sk}_{2d} \\ \mathrm{sk}_{2d+1} \\ \mathrm{sk}_{2(\ell+d)} \\ \mathrm{sk}_{2(\ell+d)+1} \end{pmatrix}$$

for an (unknown) 4×4 matrix M_d^{-1} with coefficients in $\mathbb{F}_q$ from which we learn (similar to Equation (13))

$$(\frac{\theta}{\overline{\theta}} + 1) c_i x_i^d = \delta_{(d)0} (\gamma_{(d)1} P)^{(i)} + \delta_{(d)1} (\gamma_{(d)2} P)^{(i)} + \delta_{(d)2} (\gamma_{(d)3} P)^{(i)} + \delta_{(d)3} (\gamma_{(d)4} P)^{(i)} \qquad (15)$$

for unknowns $\delta_{(d)i} \in \mathbb{F}_{q^2}$ (and unknowns x_i). How to solve such a system? Here, the freedom of choice in d allows us to choose $1 \leq d \leq t - \ell$ as a power of two. In this case, Equations (15) become *linear* in the bits of x_i when viewed as binary equations for a fixed guess for c_i. Let n_d be the number of possible choices for d, i.e. $n_d = \lfloor \log_2(t - \ell) \rfloor$. We get a linear system with $(\log_2 q^2)(4n_d + b)$ unknowns ($4n_d$ for the unknowns $\delta_{(d)i}$ and b unknowns for the points $x_{\ell j} = y_j$) and $(\log_2 q^2) n_d b$ equations ($\log_2 q^2$ equation for each d and each component $i = j\ell$). Thus whenever $b > 4$ and $n_d \geq 2$ (i.e. $t \geq 4$) this system is likely to be over defined and thus reveals the secret values x_i. We verified the behavior of the system and observed the following.

Experimental Observation 3 *Only for the right guess for the constants c_i the system is solvable. When we fix wlog $x_0 = 1$, for the right constants there is a unique solution for the values x_i.*

As there are q^6 possibilities for the constants and it takes roughly $(n_d b)(4n_d + b)^2 (\log_2 q^2)^3$ binary operations to solve the system, the overall running time of this attack is $q^6 \times (n_d b)(4n_d + b)^2 (\log_2 q^2)^3$. For the concrete parameters the attack complexity is summarized in Table 1.

5.2 Practical Attacks for parameter sets VI and VII

In this part we describe how, using Gröbner basis techniques, we can recover the secret key for the parameter sets VI and VII of Table 1 within a few seconds on a standard PC. The attack resembles in large parts the attack described above. The main difference in the solve phase is that we are not going to guess the constants to get linear equations for the points, but instead solve a non-linear system with the help of Gröbner basis techniques.

Isolate: Again, we make use of polynomials $g_\gamma = x^d$ but this time with the restriction $t - \ell \leq d < \ell$. To recover the corresponding vectors γ we make use of the space U_d defined by Equation (14). Now, with the given restriction on d it turns out that the situation, from an attacker's point of view, is nicer as for $\Gamma_d = \{\gamma \mid \gamma P \in U_d\}$, we obtain

Experimental Observation 4 *For $t - \ell \leq d < \ell$ the dimension of the space Γ_d is always* 2.

Thus, we isolated the polynomials $g(x) = \alpha_d x^d$ in this case. In other words

$$\{g_\gamma \mid \gamma \in \Gamma_d\} = \{\alpha x^d \mid \alpha \in \mathbb{F}_{q^2}\}.$$

The reason why we did not get the second term, i.e. $x^{d+\ell}$ in this case, is that the degree of g_γ is bounded by $t-1$ and $d+\ell$ exceeds this bound.

Collect Phase: Denote by $\gamma_{(d)1}, \gamma_{(d)2}$ a basis of the two dimensional space Γ_d. Referring to Equation (11) we get

$$M_d^{-1} \begin{pmatrix} \gamma_{(d)1} \\ \gamma_{(d)2} \end{pmatrix} P = \begin{pmatrix} \mathrm{sk}_{2d} \\ \mathrm{sk}_{2d+1} \end{pmatrix},$$

for an (unknown) 2×2 matrix M_d^{-1} with coefficients in $\mathbb{F}_q$.

Solve Phase: We denote the entries of M_d^{-1} by (β_{ij}). The i.th component of the first row can be rewritten as

$$\beta_{00}(\gamma_{(d)1}P)^{(i)} + \beta_{01}(\gamma_{(d)2}P)^{(i)} = c_i x_i^d + \overline{c_i x_i^d} \tag{16}$$

Again, we can assume $x_0 = c_0 = 1$. This (for $i = 0$) reveals $\beta_{00}(\gamma_{(d)1}P)^{(0)} + \beta_{01}(\gamma_{(d)2}P)^{(0)} = 0$ and thus $\beta_{01} = \frac{\beta_{00}(\gamma_{(d)1}P)^{(0)}}{(\gamma_{(d)2}P)^{(0)}}$. Substituting back into Equation (16) we get

$$\beta_{00}\left((\gamma_{(d)1}P)^{(i)} + \frac{(\gamma_{(d)1}P)^{(0)}}{(\gamma_{(d)2}P)^{(0)}}(\gamma_{(d)2}P)^{(i)}\right) = c_i x_i^d + \overline{c_i x_i^d}.$$

For parameter sets VI and VII we successfully solved this set of equations within seconds on a standard PC using MAGMA [4]. For parameters VI, d ranges from 33 to 92 and for parameters VII from 15 to 92. Thus in both cases we can expect to get a highly overdefined system. This allows us to treat $\overline{c_i}$ and $\overline{x_i^d}$ as independent variables, speeding up the task of computing the Gröbner basis by a large factor. The average running times are summarized in Table 1.

This attack does not immediately apply to parameters I to IV as here the range of d fulfilling $t-\ell \leq d < \ell$ is too small (namely $d \in \{49, 50\}$) which does not result in sufficiently many equations. However, we anticipate that using Gröbner basis techniques might speed up the attack for those parameters as well.

6 Applying the Framework to the Dyadic Variant

In this section we introduce, in a very similar way as we did in Section 5.1, how to apply the general framework of the attack to the McEliece variant introduced in [2] and described in Section 3.2. For $u = \log_2 t$ the attack to be described a complexity of roughly $q^2 \times (\log_2 q^2)^3(u^2 + 3u + b)^2 u(u + b)$ binary operations, which for the parameters given in [2] means that we can recover the secret key within at most a few days with a standard PC (cf. Table 1 for actual values).

Recovering Constants c_j

Isolate phase: As before we consider $g_\gamma(x) = 1$ and we want to compute the corresponding vector γ. From Equation (9) we have that

$$\gamma P = (\phi(c_0 g_\gamma(x_0)), \dots, \phi(c_{n-1} g_\gamma(x_{n-1}))) = (\phi(c_0), \phi(c_1), \dots, \phi(c_{n-1})).$$

Now, taking Equation (7) into account, this becomes

$$\begin{aligned}\gamma P = (&\phi(a_0), \phi(a_0), \phi(a_0) \dots, \phi(a_0),\\ &\phi(a_1), \phi(a_1), \phi(a_1) \dots, \phi(a_1),\\ &\vdots \quad\quad \vdots\\ &\phi(a_{b-1}), \phi(a_{b-1}), \phi(a_{b-1}) \dots, \phi(a_{b-1}))\,.\end{aligned}$$

We see that γ corresponding to the constant polynomial g_γ fulfils

$$\gamma P = \phi(a_0)v_0 + \phi(a_1)v_1 + \cdots + \phi(a_{b-1})v_{b-1}$$

where

$$v_i = (\underbrace{0, \dots, 0}_{i\ell}, 1, 1, 1, \dots, 1, \underbrace{0, \dots 0}_{((b-1)-i)\ell}) \quad \text{for } 0 \le i \le b-1.$$

Let U be the space spanned by v_0 up to v_{b-1}. The γ that we are looking for is such that

$$\gamma P \in U = \langle v_0, \dots, v_{b-1} \rangle.$$

Thus in order to find γ we have to compute a basis for the space $\Gamma_0 = \{\gamma \mid \gamma P \in U\}$. We did this for many randomly generated public keys and observe the following.

Experimental Observation 5 *The dimension of the space Γ_0 is always* 2.

The next lemma shows, why the dimension is at least 2.

Lemma 3. *Let γ be a vector such that $g_\gamma(x) = \alpha_0$. Then $\gamma \in \Gamma_0$.*

Note that $\dim \Gamma_0 = 2$ is actually the best case we can hope for within our framework.

Collect Phase: Denote by γ_1, γ_2 a basis of the two dimensional space Γ_0. Referring to Equation (11) we get

$$M^{-1} \begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix} P = \begin{pmatrix} \mathrm{sk}_0 \\ \mathrm{sk}_1 \end{pmatrix} \tag{17}$$

for an (unknown) 2×2 matrix M^{-1} with coefficients in $\mathbb{F}_q$.

Solve Phase: We denote the entries of M^{-1} by (β_{ij}). We get

$$\begin{pmatrix} \beta_{00}\, \beta_{01} \\ \beta_{10}\, \beta_{11} \end{pmatrix} \begin{pmatrix} \gamma 1 \\ \gamma 2 \end{pmatrix} P = \begin{pmatrix} \phi(c_0), & \phi(c_1), & \cdots, & \phi(c_{b-1}) \\ \phi(\theta c_0), & \phi(\theta c_1), & \cdots, & \phi(\theta c_{b-1}) \end{pmatrix}.$$

Assuming wlog that $c_0 = 1$, we can compute β_{01} as a function of β_{00} and β_{11} as a function of β_{10}. Then guessing β_{00} and β_{10} allows us to recover all the constants. We conclude that there are q^2 possible choices for the b constants $a_0, \dots, a_{b-1}$. We will have to repeat the following step for each of those choices.

Recovering Points $\boldsymbol{x_i}$ Assuming that we know the constants c_i we explain how to recover the secret values x_i by solving an (over-defined) system of linear equations. If the set of constants that we have chosen in the previous step is not the correct one, the system will not be solvable.

Isolate: We start by considering $g_\gamma(x) = x$, and multiply the desired vector γ with the public key P. We expect (cf. Equation (9)) to obtain the following:

$$\gamma P = (\phi(c_0 g_\gamma(x_0)), \dots, \phi(c_{n-1} g_\gamma(x_{n-1})))$$

then

$$\begin{array}{rcccc} \gamma P = & (\phi(a_0x_0), & \phi(a_0x_1), & \dots, & \phi(a_0x_{\ell-1}), \\ & \phi(a_1x_\ell), & \phi(a_1x_{\ell+1}), & \dots, & \phi(a_1x_{2\ell-1}), \\ & \vdots & \vdots & & \vdots \\ & \phi(a_{b-1}x_{(b-1)\ell}), & \phi(a_{b-1}x_{(b-1)\ell+1}), & \dots, & \phi(a_{b-1}x_{b\ell-1})). \end{array} \tag{18}$$

Recalling Equation (8) we see that the vector γ we are looking for fulfils

$$(\gamma P)^{(\ell i+j)} = \sum_{f=0}^{u-1} j_f (\gamma P)^{(\ell i + 2^f)} + (1 + W_H(j))(\gamma P)^{(\ell i)} \quad \forall\ 0 \le i < b, 0 \le j < \ell \tag{19}$$

where $j = \sum_{f=0}^{u-1} j_f 2^f$ is the binary representation of j. Denoting $\mathit{\Gamma}_1 = \{\gamma \in \mathbb{F}_q^{2t} \mid \gamma \text{ fulfils (19)}\}$ we got the following observation by randomly generating many keys.

Experimental Observation 6 *The dimension of the space $\mathit{\Gamma}_1$ is always $u+1$.*

Clearly, the dimension is at least $u+1$ as we are actually only checking if g_γ is $\mathbb{F}_2$ affine and therefore if γ is such that $g_\gamma(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \dots + \alpha_u x^{2^{u-1}}$ then $\gamma \in \mathit{\Gamma}_1$.

Collect Phase: A straight-forward application of Equation (10) would lead to a linear system that becomes only over-defined for a large number of blocks. Thus, in order to avoid this we modify the collect phase as follows. Let $\gamma \in \mathit{\Gamma}_1$ be given. We have

$$\begin{aligned} \gamma P = (&\phi(a_0 g_\gamma(x_0)), \phi(a_0 g_\gamma(x_1)), \dots, \phi(a_0 g_\gamma(x_{l-1})), \\ &\phi(a_1 g_\gamma(x_\ell)), \phi(a_1 g_\gamma(x_{\ell+1})), \dots, \phi(a_1 g_\gamma(x_{2\ell-1})), \dots) \end{aligned}$$

where g_γ is an $\mathbb{F}_2$ affine polynomial. Making use of the identity

$$x_0 + x_i = x_\ell + x_{\ell+i} \quad \forall\ 0 \le i < \ell$$

allows us to compute $\mu_\gamma^{(i)} = \phi(a_0(g_\gamma(x_0 + x_i) + g(0)))$ and $\nu_\gamma^{(i)} = \phi(a_1(g_\gamma(x_0 + x_i) + g(0)))$. As we assume we know the constants a_0 and a_1, given $\mu_\gamma^{(i)}$ and $\nu_\gamma^{(i)}$ we can recover (cf. Equation (2)) $z_\gamma^{(i)} = g_\gamma(x_0 + x_i) + g(0)$ (as long as (a_0, a_1) is an $\mathbb{F}_q$ basis of $\mathbb{F}_{q^2}$). Next, by solving a system of linear equations, we compute a γ' such that

$$z_{\gamma'}^{(i)} = \theta z_\gamma^{(i)}.$$

It turns out that the corresponding polynomial $g_{\gamma'}$ is unique up to adding constants, i.e. $g_{\gamma'} = \theta g_\gamma + c$. Summarizing our findings so far we get

$$\begin{aligned}\gamma P &= (\phi(a_0 g_\gamma(x_0)), \phi(a_0 g_\gamma(x_1)), \dots, \phi(a_{b-1} g_\gamma(x_{n-1})))\\ \gamma' P &= (\phi(\theta a_0 g_\gamma(x_0) + a_0 c), \phi(\theta a_0 g_\gamma(x_1) + a_0 c), \dots, \phi(\theta a_{b-1} g_\gamma(x_{n-1}) + a_{b-1} c)).\end{aligned}$$

This, again using Equation (2), allows us to compute

$$\delta = (a_0 g_\gamma(x_0), a_0 g_\gamma(x_1), \dots, a_{b-1} g_\gamma(x_{n-1})) + (a_0 c', a_0 c', \dots, a_{b-1} c') + (\overline{a_0} c'', \overline{a_0} c'', \dots, \overline{a_{b-1}} c'')$$

for (unknown) constants c', c''. Repeating this procedure for different elements $\gamma \in \Gamma_1$ will eventually result in $\delta_1, \dots, \delta_{u+2}$ that span a space of dimension $u+2$. The data we collected can thus be written as

$$\begin{pmatrix} \delta_1 \\ \vdots \\ \delta_{u+2} \end{pmatrix} = M \begin{pmatrix} (a_0, a_0, \dots, a_{b-1}) \\ (\overline{a_0}, \overline{a_0}, \dots, \overline{a_{b-1}}) \\ (a_0 x_0, a_0 x_1, \dots, a_{b-1} x_{n-1}) \\ \vdots \quad \vdots \\ (a_0 x_0^{2^{u-1}}, a_0 x_1^{2^{u-1}}, \dots, a_{b-1} x_{n-1}^{2^{u-1}}) \end{pmatrix} \tag{20}$$

for an invertible $(u+2) \times (u+2)$ matrix M.

Solve Phase: Multiplying Equation (20) by M^{-1} yields equations that, when viewed as binary equations, are linear in the entries of M^{-1} and the values x_i (as we assume the a_i to be known). The first two rows of M are determined by the (known) values of the constants a_i. Thus we are left with $N_u = \log_2(q^2)(u(u+2) + (u+b))$ unknowns, i.e. the remaining $u(u+2)$ entries of M^{-1} and the $u+b$ points

$$x_0, x_1, x_2, x_4, \dots, x_{2^{u-1}}, x_\ell, x_{2\ell}, x_{3\ell}, \dots x_{(b-1)\ell}$$

(all other points are given as linear combinations of those). The number of equations is $N_e = \log_2(q^2)(u+b) \times u$. In particular, whenever $b \geq 4$ and $u \geq 4$, i.e. $t \geq 2^4$, we get more equations than unknowns and can hope for a unique solution. We implemented the attack and observed the following.

Experimental Observation 7 *Only for the right guess for the constants c_i the system is solvable. In this case the constants x_0 and x_1 could be chosen as arbitrary non-zero elements in $\mathbb{F}_{q^2}$.*

As there are q^2 possibilities for the constants and it takes roughly $(N_e N_u^2)$ binary operations to solve the system, the overall running time of this attack is $q^2 \times (\log_2 q^2)^3 (u^2 + 3u + b)^2 u(u+b)$ binary operations. In Table 2 we computed the complexity of the attack for the sample parameters given in [2, Table 5].

Acknowledgement

We like to thank the authors of [3] for providing us a sample implementation of their cipher. We also like to thank Søren Thomsen and Tom Høholdt for fruitful discussions.

References

1. M. Baldi and F. Chiaraluce. Cryptanalysis of a new instance of McEliece cryptosystem based on QC-LDPC Codes. *IEEE International Symposium on Information Theory*, pages 2591–2595, 2007.
2. Paulo S. L. M. Barreto and Rafael Misoczki. Compact McEliece Keys from Goppa Codes. In *Proceedings of the 16th International Workshop on Selected Areas in Cryptography, SAC 2009*, volume 5867 of *Lecture Notes in Computer Science*. Springer-Verlag, 2009.
3. Thierry P. Berger, Pierre-Louis Cayrel, Philippe Gaborit, and Ayoub Otmani. Reducing Key Length of the McEliece Cryptosystem. In Bart Preneel, editor, *AFRICACRYPT*, volume 5580 of *Lecture Notes in Computer Science*, pages 77–97. Springer, 2009.
4. Wieb Bosma, John J. Cannon, and Catherine Playoust. The Magma Algebra System I: The User Language. *J. Symb. Comput.*, 24(3/4):235–265, 1997.
5. Thomas Eisenbarth, Tim Güneysu, Stefan Heyse, and Christof Paar. Microeliece: Mceliece for embedded devices. In Christophe Clavier and Kris Gaj, editors, *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2009.
6. J.-C. Faugère, A. Otmani, L. Perret, and J.-P. Tillich. Algebraic cryptanalysis of McEliece variants with compact keys. In *Proceedings of EUROCRYPT 2010*, LNCS. Springer, 2010. to appear.
7. P. Gaborit. Shorter keys for code based cryptography. International Workshop on Coding and Cryptography, Bergen, Norway, 2005.
8. F. J. MacWilliams and N. J. Sloane. *The theory of error-correcting codes.* North Holland, Amsterdam, 1977.
9. Robert .J. McEliece. A public key cryptosystem based on alegbraic coding theory. *DSN progress report*, 42-44:114–116, 1978.
10. C. Monico, J. Rosenthal, and A. Shokrollahi. Using low density parity check codes in the McEliece cryptosystem. *IEEE International Symposium on Information Theory*, page 215, 2000.
11. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *IEEE Symposium on Foundations of Computer Science*, pages 124–134, 1994.
12. V. M. Sidelnikov and S. O. Shestakov. On cryptosystem based on gerenalized reed-solomon codes. *Discrete Mathematics*, 4(3):57–63, 1992.
13. Mark Weiser. The computer for the 21st century. *Scientific American*, Special Issue on Communications, Computers, and Networks September, September 1991.
14. C. Wieschebrink. Two NP-complete problems in coding theory with an application in code based cryptography. *IEEE International Symposium on Information Theory*, pages 1733–1737, 2006.

A Proof of Proposition 1

Proof. For the proof it is enough to consider the effect of multiplying a vector $s \in \mathbb{F}_q^t$ by H. For convenience we label the coordinates of s as

$$s = (\alpha_0, \beta_0, \alpha_1, \beta_1, \dots, \alpha_{t-1}, \beta_{t-1})$$

We compute

$$\begin{aligned}
sH &= s \begin{pmatrix} \phi(\theta c_0) & \dots & \phi(\theta c_{n-1}) \\ \phi(c_0) & \dots & \phi(c_{n-1}) \\ \vdots & & \vdots \\ \phi(\theta c_0 x_0^{t-1}) & \dots & \phi(\theta c_{n-1} x_{n-1}^{t-1}) \\ \phi(c_0 x_0^{t-1}) & \dots & \phi(c_{n-1} x_{n-1}^{t-1}) \end{pmatrix} \\
&= \left(\sum_{i=0}^{t-1} \alpha_i \phi(\theta c_0 x_0^i) + \sum_{i=0}^{t-1} \beta_i \phi(c_0 x_0^i), \dots, \sum_{i=0}^{t-1} \alpha_i \phi(\theta c_{n-1} x_{n-1}^i) + \sum_{i=0}^{t-1} \beta_i \phi(c_{n-1} x_{n-1}^i) \right) \\
&= \left(\phi(c_0 \sum_{i=0}^{t-1} (\theta \alpha_i + \beta_i) x_0^i), \dots, \phi(c_{n-1} \sum_{i=0}^{t-1} (\theta \alpha_i + \beta_i) x_{n-1}^i) \right) \\
&= (\phi(c_0 g(x_0)), \dots, \phi(c_{n-1} g(x_{n-1})))
\end{aligned}$$

where $g(x) = \sum_{i=0}^{t-1}(\theta\alpha_i + \beta_i)x^i$. □

B A practical attack for parameter set V

Recall that β is an element of order ℓ in $\mathbb{F}_{q^2}$. Attacks for the case that β is actually in the subfield $\mathbb{F}_q$ are discussed in Section 5. In the case that β is not in the subfield things are a little different and we focus on this case here.

Isolate Phase: Assume that again we would like to isolate the polynomial $g_\gamma(x) = x^d$. Multiplying the vector γ with the public key P yields

$$\begin{aligned}\gamma P = \Big(&\phi(a_0y_0), \phi(\beta^{s+d}a_0y_0), \phi(\beta^{2(s+d)}a_0y_0)\ldots, \phi(\beta^{(\ell-1)(s+d)}a_0y_0),\\ &\phi(a_1y_1), \phi(\beta^{s+d}a_1y_1), \phi(\beta^{2(s+d)}a_1y_1)\ldots, \phi(\beta^{(\ell-1)(s+d)}a_1y_1),\\ &\ \vdots \quad \vdots\\ &\phi(a_{b-1}y_{b-1}), \phi(\beta^{s+d}a_{b-1}y_{b-1}), \ldots, \phi(\beta^{(\ell-1)(s+d)}a_{b-1}y_{b-1})\Big).\end{aligned}$$

However, as β is not in the subfield we cannot continue as before. Instead $(\gamma P)^{(0)}$ and $(\gamma P)^{(1)}$ allow to recover a_0y_0 by means of $(\gamma P)^{(0)} = \phi(a_0y_0)$ and $(\gamma P)^{(1)} = \phi(\beta^{s+d}a_0y_0)$ using Equation (2), which reveals a_0y_0 as

$$a_0y_0 = \frac{(\gamma P)^{(0)}\overline{\beta^{s+d}} + (\gamma P)^{(1)}}{\beta^{s+d}+1}.$$

The same argument reveals a_jy_j using $(\gamma P)^{(j\ell)}$ and $(\gamma P)^{(j\ell+1)}$. Therefore, when looking for γ corresponding to x^d we can solve for all γ such that γP fulfils

$$(\gamma P)^{(j\ell+i)} = \phi\left(\beta^{i(s+d)}\frac{(\gamma P)^{(j\ell)}\overline{\beta^{s+d}} + (\gamma P)^{(j\ell+1)}}{\beta^{s+d}+1}\right) \tag{21}$$

for $0 \leq j < b$ and $0 \leq i < \ell$. We denote by Γ_d the space of all possible solutions, i.e.

$$\Gamma_d = \{\gamma \mid \gamma P \text{ fulfils Equation (21) }\}$$

Experimental Observation 8 *The dimension of Γ_d is in $\{4, 6, 8\}$.*

We next explain those dimensions.

Lemma 4. *$\{g_\gamma \mid \gamma \in \Gamma_d\}$ contains all polynomials*

$$\alpha_0x^d + \alpha_1^{d+\ell} + \alpha_2x^r + \alpha_4x^{r+\ell}$$

of degree at most $t-1$ where $r = q(d+s) - s \bmod \ell$.

Proof. For this we first claim that any polynomial fulfilling either $g(\beta x) = \beta^d g(x)$ or $\beta^s g(\beta x) = \overline{\beta^{d+s}}g(x)$ is in the set. The first condition is obvious and the second follows from the fact that in this case (using Equation (1))

$$\phi(\beta^s g(\beta x)) = \phi(\overline{\beta^{d+s}}g(x)) = \phi(\beta^{d+s}\overline{g(x)})$$

and

$$\phi(g(x)) = \phi(\overline{g(x)}).$$

If $g(x)$ is a monomial $g(x) = x^r$ we get

$$g(\beta x) = \beta^r g(x)$$

Thus, to fulfil the second equations r has to fulfil.

$$r = q(d+s) - s \bmod \ell$$

□

Clearly, the smaller the dimension of Γ_d is, the better the attack. We pick only those d such that $\dim \Gamma_d = 4$ (avoiding the exponents $d+\ell$ and $r+\ell$). The condition for this is

$$t - \ell \leq d \leq \ell \text{ and } r - \ell \leq d \leq \ell$$

and $\beta^{d+s} \notin \mathbb{F}_q$. In this case

$$\{g_\gamma \mid \gamma \in \Gamma_d\} = \{\alpha_0 x^d + \alpha_1 x^r\}$$

where $r = q(d+s) - s \bmod \ell$. For parameter set V, we ran through all possible values s and verified that in any case the number of suitable exponents d is at least 8.

Collect Phase: The collect phase, too, is different in this case. Denote by $\gamma_{(d)1}$, $\gamma_{(d)2}$ two linearly independent elements in Γ_d. Define

$$g_{\gamma_{(d)1}} = \alpha_0 x^d + \alpha_1 x^r$$

and

$$g_{\gamma_{(d)2}} = \alpha'_0 x^d + \alpha'_1 x^r.$$

We have

$$\begin{aligned}(\gamma_{(d)1}P)^{(i\ell)} &= \phi(a_i g(y_i)) \\ &= \phi(a_i(\alpha_0 y_i^d + \alpha_1 y_i^r)) \\ &= \phi(a_i\alpha_0 y_i^d + \overline{a_i\alpha_1 y_i^r})\end{aligned}$$

and

$$\begin{aligned}(\gamma_{(d)1}P)^{(i\ell+1)} &= \phi(a_i\beta^s g(\beta y_i)) = \phi(a_i\beta^s(\alpha_0\beta^d y_i^d + \alpha_1\beta^r y_i^r)) \\ &= \phi(\beta^{s+d} a_i\alpha_0 y_i^d + \overline{\beta^{s+r} a_i\alpha_1 y_i^r})) \\ &= \phi(\beta^{s+d}(a_i\alpha_0 y_i^d + \overline{a_i\alpha_1 y_i^r}))\end{aligned}$$

where we made use of the identity $\overline{\beta^{s+r}} = \beta^{s+d}$. Thus, given $(\gamma_{(d)1}P)^{(i\ell)}$ and $(\gamma_{(d)1}P)^{(i\ell+1)}$ allows us to compute

$$\eta_i = a_i\alpha_0 y_i^d + \overline{a_i\alpha_1 y_i^r}$$

and similarly

$$\eta'_i = a_i\alpha'_0 y_i^d + \overline{a_i\alpha'_1 y_i^r}.$$

We obtain vectors $\eta, \eta' \in \mathbb{F}_{q^2}^b$ such that

$$\begin{pmatrix} \eta \\ \eta' \end{pmatrix} = \begin{pmatrix} \alpha_0 & \overline{\alpha_1} \\ \alpha_0' & \overline{\alpha_1'} \end{pmatrix} \begin{pmatrix} a_0 y_0^d, a_1 y_1^d, \ldots, a_{b-1} y_{b-1}^d \\ \overline{a_0 y_0^r}, \overline{a_1 y_1^r}, \ldots, \overline{a_{b-1} y_{b-1}^r} \end{pmatrix}$$

Stated differently, there exist elements $\beta_0, \beta_1, \beta_2, \beta_3$ such that

$$\begin{pmatrix} \beta_0 & \beta_1 \\ \beta_2 & \beta_3 \end{pmatrix} \begin{pmatrix} \eta \\ \eta' \end{pmatrix} = \begin{pmatrix} a_0 y_0^d, a_1 y_1^d, \ldots, a_{b-1} y_{b-1}^d \\ \overline{a_0 y_0^r}, \overline{a_1 y_1^r}, \ldots, \overline{a_{b-1} y_{b-1}^r} \end{pmatrix}. \tag{22}$$

Solve Phase: We only consider the first row of Equation (22). In other words

$$\beta_0 \eta^{(i)} + \beta_1 \eta'^{(i)} = a_i y_i^d.$$

Again, we assume wlog that $a_0 = y_0 = 1$ and this allows us to represent β_1 in terms of the unknown β_0. Thus, we finally get equations

$$\beta_0 \eta^{(i)} + \left(\frac{\beta_0 \eta^{(0)} + 1}{\eta'^0} \right) \eta'^{(i)} = a_i y_i^d.$$

Using the computer algebra package MAGMA this system of equations can be solved very quickly on a standard PC. We give the running time in Table 1.

Algebraic Cryptanalysis of Compact McEliece's Variants – Toward a Complexity Analysis

Jean-Charles Faugère[1], Ayoub Otmani[2,3], Ludovic Perret[1], and Jean-Pierre Tillich[2]

[1] SALSA Project - INRIA (Centre Paris-Rocquencourt)
UPMC, Univ Paris 06 - CNRS, UMR 7606, LIP6
104, avenue du Président Kennedy 75016 Paris, France
jean-charles.faugere@inria.fr, ludovic.perret@lip6.fr
[2] SECRET Project - INRIA Rocquencourt
Domaine de Voluceau, B.P. 105 78153 Le Chesnay Cedex - France
ayoub.otmani@inria.fr, jean-pierre.tillich@inria.fr
[3] GREYC - Université de Caen - Ensicaen
Boulevard Maréchal Juin, 14050 Caen Cedex, France.

Abstract. A new algebraic approach to investigate the security of the McEliece cryptosystem has been proposed by Faugère-Otmani-Perret-Tillich in Eurocrypt 2010. This paper is an extension of this work. The McEliece's scheme relies on the use of error-correcting codes. It has been proved that the private key of the cryptosystem satisfies a system of bi-homogeneous polynomial equations. This property is due to the particular class of codes considered which are alternant codes. These highly structured algebraic equations allowed to mount an efficient key-recovery attack against two recent variants of the McEliece cryptosystems that aim at reducing public key sizes by using quasi-cyclic or quasi-dyadic structures. Thanks to a very recent development due to Faugère-Safey el Din-Spaenlehauer on the solving of bihomogeneous bilinear systems, we can estimate the complexity of the FOPT algebraic attack. This is a first step toward providing a concrete criterion for evaluating the security of future compact McEliece variants.

Keywords : public-key cryptography, McEliece cryptosystem, algebraic cryptanalysis, F_5, bi-linear systems,.

1 Introduction

One of the main goals of the public-key cryptography is the design of secure encryption schemes by exhibiting one-way trapdoor functions. This requires the identification of supposedly hard computational problems. Although many hard problems exist and are proposed as a foundation for public-key primitives, those effectively used are essentially classical problems coming from number theory: integer factorization (e.g. in RSA) and discrete logarithm (e.g. in Diffie-Hellman key-exchange). However, the lack of diversity in public key cryptography is a major concern in the field of information security. This situation would worsen if ever quantum computers appear because schemes that are based on these classical number theory problems would become totally insecure.

Consequently, the task of identifying alternative hard problems that are not based on number theory ones constitutes a major issue in the modern public-key cryptography. Among those problems, the intractability of decoding a random linear code [7] seems to offer the most promising solution thanks to McEliece who first proposed in [24] a public-key cryptosystem based on irreducible binary Goppa codes. The class of Goppa codes represents one of the most important example of linear codes having an efficient decoding algorithm [8, 27]. The resulting cryptosystem has then very fast encryption and decryption functions [10]. A binary Goppa code is defined by a polynomial $g(z)$ of degree $r \geq 1$ with coefficients in some extension $\mathbb{F}_{2^m}$ of degree $m > 1$ over $\mathbb{F}_2$, and a n-tuple $\mathscr{L} = (x_1, \ldots, x_n)$ of distinct elements in $\mathbb{F}_{2^m}$ with $n \leq 2^m$. The trapdoor of the McEliece public-key scheme consists of the randomly picked $g(z)$ with $\mathscr{L}$ which together provide all the information to decode efficiently. The public key is a randomly picked generator matrix of the chosen Goppa code. A ciphertext is obtained by multiplying a plaintext with the public generator matrix and adding a random error vector of prescribed Hamming weight. The receiver decrypts the message thanks to the decoding algorithm that can be derived from the secrets.

After more than thirty years now, the McEliece cryptosystem still belongs to the very few public key cryptosystems which remain unbroken. Its security relies upon two assumptions: the *intractability of decoding random linear codes* [7], and the *difficulty of recovering the private key* or an equivalent one. The problem of decoding an unstructured code is a long-standing problem whose most effective algorithms [20, 21, 29, 12, 9] have an exponential time complexity. On the other hand no significant breakthrough has been observed during the past years regarding the problem of recovering the private key. Indeed, although some weak keys have been identified in [22], the only known key-recovery attack is the exhaustive search of the secret polynomial $\Gamma(z)$ of the Goppa code, and applying the *Support Splitting Algorithm* (SSA) [28] to check whether the Goppa code candidate is *permutation-equivalent* to the code defined by the public generator matrix.

Despite its impressive resistance against a variety of attacks and its fast encryption and decryption, McEliece cryptosystem has not stood up to RSA for practical applications. This is most likely due to the large size of the public key which is between several hundred thousand and several million bits. To overcome this limitation, a trend had been initiated in order to decrease the key size by focusing on very structured codes. For instance, quasi-cyclic code like in [19], or quasi-cyclic codes defined by sparse matrices (also called LDPC codes) [1]. Both schemes were broken in [26]. It should be noted that the attacks have no impact on the security of the McEliece cryptosystem since both proposals did not use the binary Goppa codes of the McEliece cryptosystem. These works were then followed by two independent proposals [6, 25] that are based on the same kind of idea of using quasi-cyclic [6] or quasi-dyadic structure [25]. These two approaches were also broken in [18] where for the first time an algebraic attack is introduced against the McEliece cryptosystem.

Algebraic cryptanalysis is a general framework that permits to assess the security of theoretically all cryptographic schemes. So far, such type of attacks has been applied successfully against several multivariate schemes and stream ciphers. The basic principle of this cryptanalysis is to associate to a cryptographic primitive a set of algebraic equations. The system of equations is constructed in such a way to have a correspondence between the solutions of this system, and a secret information of the cryptographic primitive (for instance the secret key of an encryption scheme). In the case of the McEliece cryptosystem, the algebraic system that has to be solved has the following very specific structure:

$$\mathsf{McE}_{k,n,r}(\mathbf{X},\mathbf{Y}) = \left\{ g_{i,0}Y_0X_0^j + \cdots + g_{i,n-1}Y_{n-1}X_{n-1}^j = 0 \;\middle|\; i \in \{0,\ldots,k-1\}, j \in \{0,\ldots,r-1\} \right\} \tag{1}$$

where the unknowns are the X_i's and the Y_i's and the $g_{i,j}$'s are known coefficients with $0 \le i \le k-1, 0 \le j \le n-1$ that belong to a certain field $\mathbb{F}_q$ with $q = 2^s$. We look for solutions of this system in a certain extension field $\mathbb{F}_{q^m}$. Here k is an integer which is at least equal to $n - rm$. By denoting $\mathbf{X} \overset{\text{def}}{=} (X_0,\ldots,X_{n-1})$ and $\mathbf{Y} \overset{\text{def}}{=} (Y_0,\ldots,Y_{n-1})$ we will refer to such an algebraic system by $\mathsf{McE}_{k,n,r}(\mathbf{X},\mathbf{Y})$. This algebraic approach as long as the codes that are considered are alternant codes. It is important to note that a Goppa code can also be seen as a particular alternant code. However, it is not clear whether an algebraic attack can be mounted efficiently against the original McEliece cryptosystem because the total number of equations is rk, the number of unknowns $2n$ and the maximum degree $r-1$ of the equations can be extremely high (e.g. $n = 1024$ and $r-1 = 49$).

But in the case of the tweaked McEliece schemes [6, 25], it turns out that is possible to make use of this structure in order to reduce considerably the number of unknowns in the algebraic system. This is because of the type of codes that are considered: quasi-cyclic alternant codes in [6] and quasi-dyadic Goppa codes in [25]. In particular, it induces an imbalance between the $\mathbf{X}$ and $\mathbf{Y}$ variables. Moreover, it was possible to solve efficiently the algebraic system thanks to a dedicated Gröbner bases techniques. Finally, it was also observed experimentally in [18] but not formally proved that the complexity of the attack is mainly determined by the number of remaining variables in the block $\mathbf{Y}$.

The motivation of this paper is to revisit the FOPT algebraic attack [18] in view of the recent results on bilinear systems [16]. This permits to make more precise the dependency between the security of a McEliece (and its variants) and the properties of the algebraic system (1). This is a first step toward providing a concrete criterion for evaluating the security of future compact McEliece variants.

Organisation of the paper. After this introduction, the paper is organized as follows. We briefly recall the McEliece cryptosystem in the next section. In Section 3, we recall how we can derive the algebraic system (1). We emphasize that these parts are similar to the ones in [18]. Section 4 is the core of the paper. We explain how we can extract a suitable (i.e. affine bi-linear) system from $\mathsf{McE}_{k,n,r}(\mathbf{X},\mathbf{Y})$. We then recall new results on the complexity of solving generic affine bi-linear systems, which permit to obtain a rough estimate of the complexity of the FOPT attack. Finally, in Section 6, we compare our theoretical bound with the practical results obtained in [18].

2 McEliece Public-Key Cryptosystem

We recall here how the McEliece public-key cryptosystem is defined.

Secret key: the triplet (S, G_s, P) of matrices defined over a finite field $\mathbb{F}_q$ over q elements, with q being a power of two, that is $q = 2^s$. G_s is a full rank matrix of size $k \times n$, with $k < n$, S is of size $k \times k$ and is invertible, and P is permutation matrix of size $n \times n$. Moreover G_s defines a code (which is the set of all possible uG_s with u ranging over $\mathbb{F}_q^k$) which has a decoding algorithm which can correct in polynomial time a set of errors of weight at most t. This means that it can recover in polynomial time u from the knowledge of $uG_s + e$ for all possible $e \in \mathbb{F}_q^n$ of Hamming weight at most t.

Public key: the matrix product $G = SG_sP$.

Encryption: A plaintext $u \in \mathbb{F}_q^k$ is encrypted by choosing a random vector e in $\mathbb{F}_q^n$ of weight at most t. The corresponding ciphertext is $c = uG + e$.

Decryption: $c' = cP^{-1}$ is computed from the ciphertext c. Notice that $c' = (uSG_sP + e)P^{-1} = uSG_s + eP^{-1}$ and that eP^{-1} is of Hamming weight at most t. Therefore the aforementioned decoding algorithm can recover in polynomial time uS. This vector is multiplied by S^{-1} to obtain the plaintext u.
This describes the general scheme suggested by McEliece. From now on, we say that G is the *public generator matrix* and the vector space $\mathscr{C}$ spanned by its rows is the *public code i.e.* $\mathscr{C} \overset{\text{def}}{=} \{uG \mid u \in \mathbb{F}_q^k\}$. What is generally referred to as the McEliece cryptosystem is this scheme together with a particular choice of the code, which consists in taking a binary Goppa code. This class of codes belongs to a more general class of codes, namely the alternant code family ([23, Chap. 12, p. 365]). The main feature of this last class of codes is the fact that they can be decoded in polynomial time.

3 McEliece's Algebraic System

In this part, we explain more precisely how we construct the algebraic system described in (1). As explained in the previous section, the McEliece cryptosystem relies on Goppa codes which belong to the class of *alternant codes* and inherit from this an efficient decoding algorithm. It is convenient to describe such codes through a *parity-check matrix*. This is an $r \times n$ matrix H defined – over an extension $\mathbb{F}_{q^m}$ of the field where the code is constructed – as follows:

$$\{uG_s \mid u \in \mathbb{F}_q^k\} = \{c \in \mathbb{F}_q^n \mid Hc^T = 0\}. \tag{2}$$

r satisfies in this case the condition $r \geq \frac{n-k}{m}$. For alternant codes, there exists a parity-check matrix with a very special form related to Vandermonde matrices. More precisely there exist two vectors $x = (x_0, \dots, x_{n-1})$ and $y = (y_0, \dots, y_{n-1})$ in $\mathbb{F}_{q^m}^n$ such that $V_r(x,y)$ is a parity-check matrix, with

$$V_r(x,y) \overset{\text{def}}{=} \begin{pmatrix} y_0 & \cdots & y_{n-1} \\ y_0x_0 & \cdots & y_{n-1}x_{n-1} \\ \vdots & & \vdots \\ y_0x_0^{r-1} & \cdots & y_{n-1}x_{n-1}^{r-1} \end{pmatrix}. \tag{3}$$

We use the following notation in what follows.

Definition 1. *The alternant code $\mathscr{A}_r(x,y)$ of order r over $\mathbb{F}_q$ associated to $x = (x_0,\ldots,x_{n-1})$ where the x_i's are different elements of $\mathbb{F}_{q^m}$ and $y = (y_0,\ldots,y_{n-1})$ where the y_i's are nonzero elements of $\mathbb{F}_{q^m}$ is defined by $\mathscr{A}_r(x,y) = \{c \in \mathbb{F}_q^n \mid V_r(x,y)c^T = 0\}$.*

It should be noted that the public code in the McEliece scheme is also an alternant code. We denote here by the public code, the set of vectors of the form

$$\{uG \mid u \in \mathbb{F}_q^k\} = \{cSG_sP \mid c \in \mathbb{F}_q^k\}.$$

This is simple consequence of the fact that the set $\{uSG_sP \mid u \in \mathbb{F}_q^k\}$ is obtained from the secret code $\{uG_s \mid u \in \mathbb{F}_q^k\}$ by permuting coordinates in it with the help of P, since multiplying by an invertible matrix S of size $k \times k$ leaves the code globally invariant. The key feature of an alternant code is the following fact.

Fact 1. *There exists a polynomial time algorithm decoding an alternant code once a parity-check matrix H of the form $H = V_r(x,y)$ is given.*

In other words, it is possible to break the McEliece scheme once we can find x^* and y^* in $\mathbb{F}_{q^m}^n$ such that

$$\{xG \mid x \in \mathbb{F}_q^n\} = \{y \in \mathbb{F}_q^n \mid V_r(x^*,y^*)y^T = 0\}. \tag{4}$$

From the knowledge of this matrix $V_r(x^*,y^*)$, it is possible to decode the public code, that is to say to recover u from $uG+e$. Finding such a matrix clearly amounts to find a matrix $V_r(x^*,y^*)$ such that $V_r(x^*,y^*)G^T = 0$. Let $X_0,\ldots,X_{n-1}$ and $Y_0,\ldots,Y_{n-1}$ be $2n$ variables corresponding to the x_i^*s and y_i^* respectively. We see that finding such values is equivalent to solve the following system:

$$\left\{ g_{i,0}Y_0X_0^j + \cdots + g_{i,n-1}Y_{n-1}X_{n-1}^j = 0 \,\middle|\, i \in \{0,\ldots,k-1\}, j \in \{0,\ldots,r-1\} \right\} \tag{5}$$

where the $g_{i,j}$'s are the entries of the known matrix G with $0 \le i \le k-1$ and $0 \le j \le r-1$.

The cryptosystems proposed in [6, 25] follow the McEliece scheme [24] with the additional goal to design a public-key cryptosystem with very small key sizes. They both require to identify alternant codes having a property that allows matrices to be represented by very few rows. In the case of [6] circulant matrices are chosen whereas the scheme [25] focuses on dyadic matrices. These two families have in common the fact the matrices are completely described from the first row. The public generator matrix G in these schemes is a block matrix where each block is circulant in [6] and dyadic in [25]. The algebraic approach previously described leaded to a key-recovery in nearly all the parameters proposed in both schemes [18]. The crucial point that makes the attack possible is due to the very particular structure of the matrices and their block form describing the public alternant codes. This permits to drastically reduce the number of variables in $\mathsf{McE}_{k,n,r}(\mathbf{X},\mathbf{Y})$.

4 On Solving $\mathsf{McE}_{k,n,r}(\mathbf{X},\mathbf{Y})$

Thanks to a very recent development [16] on the solving of bi-linear systems, we can revisit the strategy used in [18] to solve $\mathsf{McE}_{k,n,r}(\mathbf{X},\mathbf{Y})$. As we will see, this permits to evaluate the complexity of computing a Gröbner bases of $\mathsf{McE}_{k,n,r}(\mathbf{X},\mathbf{Y})$ for compact variants of McEliece such as [6, 25]. Before that, we recall basic facts about the complexity of computing Gröbner bases [11, 13–15].

4.1 General Complexity of Gröbner Bases

The complexity of computing such bases depends on the so-called *degree of regularity*, which can be roughly viewed as the maximal degree of the polynomials appearing during the computation. This degree of regularity, denoted D_{reg} in what follows, is the key parameter. Indeed, the cost of computing a Gröbner basis is polynomial in the degree of regularity D_{reg}. Precisely, the complexity is:

$$\mathscr{O}\left(\binom{N+D_{\mathrm{reg}}}{D_{\mathrm{reg}}}^{\omega}\right), \tag{6}$$

which basically correspond to the complexity of reducing a matrix of size $\binom{N+D_{\mathrm{reg}}}{D_{\mathrm{reg}}}$ ($2 < \omega \leq 3$ is the "linear algebra constant", and N the number of variables of the system). The behavior of the degree of regularity D_{reg} is well understood for random (i.e. regular and semi-regular) systems [2, 4, 3, 5].

Proposition 1. *The degree of regularity of a square regular quadratic system in* $\mathbf{X}$ *is bounded by:*

$$1 + n_X, \tag{7}$$

where n_X is the number of variables in the set of variables $\mathbf{X}$. *Consequently, the maximal degree occurring in the computation of a DRL Gröbner basis (Degree Reverse Lexicographical order see [13]) is bounded by the same bound* (7).

On the contrary, as soon as the system has some kind of structure as for $\mathsf{McE}_{k,n,r}(\mathbf{X},\mathbf{Y})$, this degree is much more difficult to predict in general. Typically, It is readily seen that $\mathsf{McE}_{k,n,r}(\mathbf{X},\mathbf{Y})$ has a very specific structure, it is bi-homogeneous (i.e. product of two homogeneous polynomials with distinct variables).

4.2 Extracting a Bi-Affine System from $\mathsf{McE}_{k,n,r}(\mathbf{X},\mathbf{Y})$

As explained, $\mathsf{McE}_{k,n,r}(\mathbf{X},\mathbf{Y})$ is highly structured. It is very sparse as the only monomials occurring in the system are of the form $Y_i X_i^j$, with $0 \leq i \leq k-1$ and $0 \leq j \leq r-1$. It can also be noticed that each block of k equations is *bi-homogeneous*, i.e. homogeneous if the variables of $\mathbf{X}$ (resp. $\mathbf{Y}$) are considered alone. More precisely, we shall say that $f \in \mathbb{F}_{q^m}[\mathbf{X},\mathbf{Y}]$ is *bi-homogeneous* of *bi-degree* (d_1, d_2) if:

$$\forall \alpha, \mu \in \mathbb{F}_{q^m},\ f(\alpha\mathbf{X}, \mu\mathbf{Y}) = \alpha^{d_1}\mu^{d_2} f(\mathbf{X}, \mu\mathbf{Y}).$$

Note that the equations occurring in $\mathsf{McE}_{k,n,r}(\mathbf{X},\mathbf{Y})$ are of bi-degree $(j,1)$, with $j, 0 \leq j \leq r-1$.

We briefly recall now the strategy followed in [18] to solve $\mathsf{McE}_{k,n,r}(\mathbf{X},\mathbf{Y})$. The first fundamental remark is that there are k linear equations in the n variables of the block $\mathbf{Y}$ in $\mathsf{McE}_{k,n,r}(\mathbf{X},\mathbf{Y})$. This implies that all the variables of the block $\mathbf{Y}$ can be expressed in terms of $n_{Y'} \geq n-k$ variables. From now on, we will always assume that the variables of the block $\mathbf{Y}'$ only refer to these $n_{Y'}$ *free* variables. The first step is then to rewrite the system (1) only in function of the variables of $\mathbf{X}$ and $\mathbf{Y}'$, i.e., the variables of $\mathbf{Y} \setminus \mathbf{Y}'$ are substituted by linear combinations involving only variables of $\mathbf{Y}'$.

In the particular cases of [6, 25], the quasi-cyclic and dyadic structures provide additional linear equations in the variables of $\mathbf{X}$ and $\mathbf{Y}'$ which can be also used to rewrite/clean the system. In the sequel, we denote by $\mathsf{McE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$ the system obtained from $\mathsf{McE}_{k,n,r}(\mathbf{X},\mathbf{Y})$ by removing all the linear equations in $\mathbf{X}$ and $\mathbf{Y}$.

This system $\mathsf{McE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$ being naturally overdetermined, we can "safely" remove some equations. In [6, 25], the system $\mathsf{McE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$ is always defined over a field of characteristic two. It makes sense then to consider the set of equations of $\mathsf{McE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$ whose degree in the variables of $\mathbf{X}'$ is a power of 2, i.e. equations of bi-degree $(2^j, 1)$. We obtain in this way a sub-system of $\mathsf{McE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$, denoted $\mathsf{BiMcE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$, having $n_{X'}$ and $n_{Y'}$ variables and at most $k \cdot \log_2(r)$ equations. This system is a "quasi" bi-linear system over $\mathbb{F}_2^m$ as $\mathsf{McE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$ viewed over $\mathbb{F}_2$ is bi-linear. Note that some constant terms can occur in $\mathsf{McE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$, so the system is more precisely *affine* bi-linear.

Proposition 2. *Let* $\mathsf{BiMcE}_{k,n,r}(\mathbf{X}',\mathbf{Y}') \subset \mathbb{F}_{q^m}[\mathbf{X}',\mathbf{Y}']$ *be the system from* $\mathsf{McE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$ *by considering only the equations of bi-degree* $(2^j, 1)$. *This system has $n_{X'} + n_{Y'}$ variables, at most $k \cdot \log_2(r)$ equations and is affine bi-linear.*

4.3 On the Complexity of Solving Affine Bi-Linear Systems

Whilst the complexity of solving general bi-homogenous system is not known, the situation is different for bi-affine (resp. bi-linear) systems. In particular, the theoretical complexity is well mastered, and there is a now a dedicated algorithm for such systems [16]. As already explained, our equations are "quasi" bi-linear as we are working with equations of bi-degree $(1, 2^j)$ over a field of characteristic 2. The results presented in [16] can be then extended with a slight adaptation to the context.

A first important result of [16] is that F$_5$ [15] algorithm is already optimal for "generic" (random) affine bi-linear systems, i.e. all reductions to zero are removed by the F$_5$ criterion. Another fundamental result is that the degree of regularity of a square generic affine bi-linear system is much smaller than the degree of regularity of a generic system. It has been proved [16] that:

Proposition 3. *The degree of regularity of a square generic affine bi-linear system in* $\mathbf{X}$*' and* $\mathbf{Y}$*' is bounded by:*

$$1+\min(n_{X'},n_{Y'}), \tag{8}$$

where $n_{X'}$ *and* $n_{Y'}$ *are the number of variables in the blocks* $\mathbf{X}'$ *and* $\mathbf{Y}'$ *respectively. Consequently, the maximal degree occurring in the computation of a DRL Gröbner basis is also bounded by* (8).

Remark 1. This bound is sharp for a generic square affine bi-linear system and is much better than the usual Macaulay's bound (7) for a similar quadratic system (that is to say a system of $n_{X'}+n_{Y'}$ quadratic equations in $n_{X'}+n_{Y'}$ variables):

$$1+\min(n_{X'},n_{Y'}) \ll 1+n_{X'}+n_{Y'}$$

Since $\mathsf{BiMcE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$ is a bilinear system it is reasonable to derive a bound for this system from the previous result:

Proposition 4. *Let* $\mathsf{BiMcE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$ *be as defined below. The maximum degree reached when computing a Gröbner basis of* $\mathsf{BiMcE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$ *is smaller that:*

$$1+\min(n_{X'},n_{Y'}).$$

Remark 2. Note that the bound is not tight at all. In our situation the affine bi-linear systems are overdetermined whilst [16] only considered systems with at most as many variables than the number of equations.

Finally, it appears [16] that the matrices occurring during the matrix version of F$_5$ can be made divided into smaller matrices thanks to the bi-linear structure. Let $\dim(R_{d_1,d_2}) = \binom{d_1+n_{X'}}{d_1}\binom{d_2+n_{Y'}}{d_2}$. More precisely, the matrices occurring at degree D during the matrix F$_5$ on a bi-linear systems are of size: $\left(\dim(R_{d_1,d_2}) - [t_1^{d_1}t_2^{d_2}]\mathrm{HS}(t_1,t_2)\right) \times \dim(R_{d_1,d_2})$ for all (d_1,d_2) such that $d_1+d_2 = D, 1 \le d_1,d_2 \le D-1$, where the notation $[t_1^{d_1}t_2^{d_2}]\mathrm{HS}(t_1,t_2)$ stands for the coefficient of the term $t_1^{d_1}t_2^{d_2}$ in the Hilbert bi-serie $\mathrm{HS}(t_1,t_2)$ defined in the appendix.

As pointed out, these results hold for a bi-linear system. For an affine bi-linear, this can be considered as a good (i.e. first order) approximation. The idea is that we have to "bi-homogenize" the affine bi-linear system which corresponds to add some columns. We can then estimate the space/time complexity of computing a Gröbner basis of $\mathsf{BiMcE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$.

Proposition 5. *Let* $D = \min(n_{X'}+1, n_{Y'}+1)$. *The time complexity of computing a DRL-Gröbner basis* G_{DRL} *of* $\mathsf{BiMcE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$ *is bounded from above by:*

$$\left(\sum_{\substack{d_1+d_2=D\\ 1\le d_1,d_2\le D-1}} \left(\dim(R_{d_1,d_2}) - [t_1^{d_1}t_2^{d_2}]\mathrm{HS}(t_1,t_2)\right)^{\omega} \dim(R_{d_1,d_2})\right), \textit{with } \omega, 1 \le \omega \le 2.$$

The space complexity is bounded by:

$$\left(\sum_{\substack{d_1+d_2=D\\ 1\le d_1,d_2\le D-1}} \left(\dim(R_{d_1,d_2}) - [t_1^{d_1}t_2^{d_2}]\mathrm{HS}(t_1,t_2)\right) \dim(R_{d_1,d_2})\right),$$

It is worth to mention that, for the cryptosystems considered in [18], the number of free variables $n_{Y'}$ in $\mathbf{Y}'$ can be rather small (typically 1 or 2 for some challenges). We have then a theoretical explanation of the practical efficiency observed in [18]. In addition, we have a concrete criteria to evaluate the security of future compact McEliece's variants, namely the minimum of the number of variables $n_{X'}$ and $n_{Y'}$ in the blocks $\mathbf{X}'$ and $\mathbf{Y}'$ respectively should be sufficiently "big". This will be further discussed in the last section.

To conclude this section, we mention that the goal of the attack is compute the variety (i.e. set of solutions) $\mathscr{V}$ associated to $\mathsf{McE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$. As soon as we have a DRL-Gröbner basis G_{DRL} of $\mathsf{BiMcE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$, the variety can be obtained in $\mathscr{O}\big((\#\mathscr{V})^{\omega}\big)$ thanks to a change of ordering algorithm [17]. We have to be sure that the variety $\mathscr{V}$ has few solutions. In particular, we have to remove parasite solutions (corresponding to $X_i = X_j$ or to $Y_j = 0$). A classical way to do that is to introduce new variables u_{ij} and v_i and add to $\mathsf{McE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$ equations of the form: $u_{ij}\cdot(X_i - X_j) + 1 =$ and $v_i \cdot Y_i + 1 = 0$. In practice, we have not added all theses equations; but only few of them (namely 4 or 5). The reason is that we do not want to add too many new variables. These equations and variables can be added to $\mathsf{BiMcE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$ whilst keeping the affine bi-linear structure. To do so, we have to add the v_i to the block $\mathbf{X}'$, and the variables u_{ij} to the block $\mathbf{Y}'$. So, as we add only few new variables, the complexity of solving $\mathsf{BiMcE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$ with these new constraints is essentially similar to Proposition 5.

5 Application to Key Recovery Attacks of Compact McEliece Variants

The algebraic approach as described in Section 3 had been applied in [18] to two variants of the McEliece cryptosystem [6, 25]. These two systems propose code-based public-key cryptosystems with compact keys by using structured matrices. The BCGO cryptosystem in [6] relies on quasi-cyclic alternant codes whereas the MB cryptosystem in [25] uses quasi-dyadic Goppa codes. The most important fact is that the introduction of structured matrices induces linear relations between the x_i's and the y_j's defining the secret code. We briefly recall how they are built and we refer the reader to [18] for more details.

In both schemes, the public code $\mathscr{C}$ is defined over a field $\mathbb{F}_q = \mathbb{F}_{2^s}$ which is considered as a subfield of $\mathbb{F}_{q^m}$ for a certain integer m.The length n and the dimension k of $\mathscr{C}$ are always of the form $n = n_0\ell$ and $k = k_0\ell$ where ℓ divides $q^m - 1$ and n_0 and k_0 are integers such that $k < n < q^m$. We now give the additional linear equations that link the x_i's and the y_i's in order to describe how the codes are obtained.

BCGO Scheme. Let α be a primitive element of $\mathbb{F}_{q^m}$. Let ℓ and N_0 be such that $q^m - 1 = N_0\ell$ and let β be an element of $\mathbb{F}_{q^m}$ of order ℓ, that is to say $\beta \overset{\mathrm{def}}{=} \alpha^{N_0}$. The public code is an alternant code $\mathscr{A}_r(x,y)$ such that $rm = n - k = (n_0 - k_0)\ell$ and where $x = (x_0,\ldots,x_{n-1})$ and $y = (y_0,\ldots,y_{n-1})$ satisfy for any $b \in \{0,\ldots,n_0 - 1\}$ and for any $j \in \{0,\ldots,\ell-1\}$ the following linear equations [18]:

$$\begin{cases} x_{b\ell+j} = x_{b\ell}\beta^{j} \\ \\ y_{b\ell+j} = y_{b\ell}\beta^{je} \end{cases} \tag{9}$$

where e is an integer secretly picked in $\{0,\ldots,\ell-1\}$. We are able to simplify the description of the system $\mathsf{McE}_{k,n,r}(\mathbf{X},\mathbf{Y})$ by setting up the unknown X_b for $x_{b\ell}$ and the unknown Y_b for $y_{b\ell}$. We obtain the following algebraic system in which we assume that e is known:

Proposition 6 ([18]). *Let $G = (g_{i,j})$ be the $k \times n$ public generator matrix with $k = k_0\ell$ and $n = n_0\ell$. For any $0 \le w \le r-1$ and any $0 \le i \le k-1$, the unknowns $X_0,\ldots,X_{n_0-1}$ and $Y_0,\ldots,Y_{n_0-1}$ should satisfy:*

$$\sum_{b=0}^{n_0-1} g'_{i,b,w} Y_b X_b^{w} = 0 \qquad \text{where } g'_{i,b,w} \overset{def}{=} \sum_{j=0}^{\ell-1} g_{i,b\ell+j}\beta^{j(e+w)}. \tag{10}$$

Furthermore, one X_i can be set to any arbitrary value (say $X_0 = 0$) as well as one Y_i can be set to any arbitrary nonzero value (say $Y_0 = 1$). Finally, The system (10) has $(n_0 - 1)$ unknowns Y_i and $(n_0 - 1)$ unknowns X_i. It has k_0 linear equations involving only the Y_i's and $(r-1)k/\ell = (r-1)k_0$ polynomial equations involving the monomials $Y_iX_i^w$ with $w > 0$.

We have then:

Corollary 1. *The system* (10) *has $n_{Y'} = n_0 - k_0 - 1$ free variables in the Y_b's.*

MB Scheme. The public code defined by the (public) generator matrix G can be seen as an alternant code $\mathscr{A}_\ell(x,y)$ (that is to say $r=\ell$) where for any $0 \le b \le n_0-1$ and $0 \le i \le \ell-1$, we have the following linear equations [18]:

$$\begin{cases} y_{b\ell+i} = y_{b\ell} \\ \\ x_{b\ell+i} = x_{b\ell} + \sum_{j=0}^{\log_2(\ell-1)} \eta_j(i)(x_{2^j}+x_0) \end{cases} \tag{11}$$

where $\sum_{j=0}^{\log_2(\ell-1)} \eta_j(i)2^j$ with $\eta_j(i) \in \{0,1\}$ is the binary decomposition of i. This description enables to simplify the unknowns involved in $\mathsf{McE}_{k,n,r}(\mathbf{X},\mathbf{Y})$ to $Y_{b\ell}$, $X_{b\ell}$ with $b \in \{0,\ldots,n_0-1\}$ and to the unknowns X_{2^j} with $j \in \{0,\ldots,\log_2(\ell-1)\}$ We then obtain the following algebraic system:

Proposition 7 ([18]). *Let $G=(g_{i,j})$ be the $k \times n$ public generator matrix with $k=k_0\ell$ and $n=n_0\ell$. For any w, i such that $0 \le w \le \ell-1$ and $0 \le i \le k-1$, we have:*

$$\sum_{b=0}^{n_0-1} Y_{b\ell} \sum_{j=0}^{\ell-1} g_{i,b\ell+j} \left(X_{b\ell} + \sum_{j=0}^{\log_2(\ell-1)} \eta_j(j)(X_{2^j}+X_0) \right)^w = 0 \tag{12}$$

Furthermore, two X_i's can be set to any (different) arbitrary values (say $X_0=0$ and $X_1=1$) as well as one Y_i can be set to any arbitrary nonzero value (say $Y_0=1$). Finally, The system (12) has n_0-1 unknowns Y_i and $n_0-2+\log_2(\ell)$ unknowns X_i. Furthermore, it has n_0-m linear equations involving only the Y_i's, and $(\ell-1)\ell(n_0-m)$ polynomial equations involving the monomials $Y_iX_i^w$ with $w>0$.

It holds then:

Corollary 2. *The system* (12) *has $n_{Y'}=m-1$ free variables in the Y_b's.*

6 Comparison of Theoretical complexity with Experimental Results

In the table below, we present the experimental results obtained in [18] for BCGO and MB schemes. For the sack of comparaison, we include a bound on theoretical complexity of computing a Gröbner bases of $\mathsf{BiMcE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$:

$$T_{\text{theo}} \approx \left(\sum_{\substack{d_1+d_2=D \\ 1 \le d_1,d_2 \le D-1}} \Big(\dim(R_{d_1,d_2}) - [t_1^{d_1}t_2^{d_2}]\mathrm{HS}(t_1,t_2)\Big) \dim(R_{d_1,d_2}) \right), \tag{13}$$

as obtained in Section 4. Regarding the linear algebra, this is a bit optimistic. However, as already pointed our, we have been also rather pessimistic regarding others parameters. For instance, we are not using the fact that the systems are overdetermined, and we have also only considered a sub-system of $\mathsf{McE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$.

All in all, this bound permits a give a reasonable picture of the hardness of solving $\mathsf{BiMcE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$. It is of course not sufficient to set parameters, but sufficient to discard many weak compact variants of McEliece.

Table 1. Cryptanalysis results for [6] ($m = 2$)

Challenge	q	ℓ	n_0	$n_{Y'}$	Security [6]	$n_{X'}$	Equations	Time (Operations, Memory)	T_{theo}
A_{16}	2^8	51	9	3	80	8	510	0.06 sec ($2^{18.9}$ op, 115 Meg)	2^{17}
B_{16}	2^8	51	10	3	90	9	612	0.03 sec ($2^{17.1}$ op, 116 Meg)	2^{18}
C_{16}	2^8	51	12	3	100	11	816	0.05 sec ($2^{16.2}$ op, 116 Meg)	2^{20}
D_{16}	2^8	51	15	4	120	14	1275	0.02 sec ($2^{14.7}$ op, 113 Meg)	2^{26}
A_{20}	2^{10}	75	6	2	80	5	337	0.05 sec ($2^{15.8}$ op, 115 Meg)	2^{10}
B_{20}	2^{10}	93	6	2	90	5	418	0.05 sec ($2^{17.1}$ op, 115 Meg)	2^{10}
C_{20}	2^{10}	93	8	2	110	7	697	0.02 sec ($2^{14.5}$ op, 115 Meg)	2^{11}
QC$_{600}$	2^8	255	15	3	600	14	6820	0.08 sec ($2^{16.6}$ op, 116 Meg)	2^{21}

Table 2. Cryptanalysis results for [25].

Challenge	q	$n_{Y'}$	ℓ	n_0	Security	$n_{X'}$	Equations	Time (Operations, Memory)	T_{theo}
Table 2	2^2	7	64	56	128	59	$193,584$	$1,776.3$ sec ($2^{34.2}$ op, 360 Meg)	2^{65}
Table 2	2^4	3	64	32	128	36	$112,924$	0.50 sec ($2^{22.1}$ op, 118 Meg)	2^{29}
Table 2	2^8	1	64	12	128	16	$40,330$	0.03 sec ($2^{16.7}$ op, 35 Meg)	2^8
Table 3	2^8	1	64	10	102	14	$32,264$	0.03 sec ($2^{15.9}$ op, 113 Meg)	2^8
Table 3	2^8	1	128	6	136	11	$65,028$	0.02 sec ($2^{15.4}$ op, 113 Meg)	2^7
Table 3	2^8	1	256	4	168	10	$130,562$	0.11 sec ($2^{19.2}$ op, 113 Meg)	2^7
Table 5	2^8	1	128	4	80	9	$32,514$	0.06 sec ($2^{17.7}$ op, 35 Meg)	2^6
Table 5	2^8	1	128	5	112	10	$48,771$	0.02 sec ($2^{14.5}$ op, 35 Meg)	2^7
Table 5	2^8	1	128	6	128	11	$65,028$	0.01 sec ($2^{16.6}$ op, 35 Meg)	2^7
Table 5	2^8	1	256	5	192	11	$195,843$	0.05 sec ($2^{17.5}$ op, 35 Meg)	2^7
Table 5	2^8	1	256	6	256	12	$261,124$	0.06 sec ($2^{17.8}$ op, 35 Meg)	2^7
Dyadic$_{256}$	2^4	3	128	32	256	37	$455,196$	7.1 sec ($2^{26.1}$ op, 131 Meg)	2^{29}
Dyadic$_{512}$	2^8	1	512	6	512	13	$1,046,532$	0.15 sec ($2^{19.7}$ op, 38 Meg)	2^8

We briefly discussed of the theoretical complexity obtained for the first row of the second column. As explained, we have used the formula (13). We have computed the coefficient $[t_1^{d_1} t_2^{d_2}]\mathrm{HS}(t_1,t_2)$ by using the explicit formula of $\mathrm{HS}(t_1,t_2)$ provided in the appendix using the explicit values of $n_{X'} = 59$ and $n_{Y'}$=7, and assuming that the system is square; in that case the degree of regularity is 8. For this parameter, the sub-system $\mathsf{BiMcE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$ has actually 288 equations (of degree $2,3$ and 5). Hence, it is interesting to compute [2, 4, 3, 5] the degree of regularity of a semi-regular system of the same size: we found a regularity of 11 leading to a cost of $2^{85.2}$ for the Gröbner basis computation (using (6), with $\omega = 2$). It is expected that a new results of the degree of regularity of generic overdetermined bi-linear systems would lead to tighter bounds.

As a conclusion, one can see that the theoretical bound (13) provides a reasonable explanation regarding the efficiency of the attack presented in [18]. In particular, it is important to remark that the hardness of the attack seems related to $d = \min(n'_X, n'_Y)$. The complexity of the attack clearly increases with this quantity. For the design of future compact variants of McEliece, this d should be then not too small. Regarding the current state of the art, it is difficult to provide an exact value. Very roughly speaking, $\mathsf{BiMcE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$ can be considered as hard as solving a random (overdetermined) algebraic system with $d = \min(n_{X'}, n_{Y'})$ equations over a big field. With this in mind, we can say that any system with $d \leq 20$ should be within the scope of an algebraic attack.

Note that another phenomena, which remains to be treated, can occur. In the particular case of binary dyadic codes, the Gröbner basis of $\mathsf{BiMcE}_{k,n,r}(\mathbf{X}',\mathbf{Y}')$ can be easily computed, but the variety associated is too big. This is due to the fact that the Gröbner basis is "trivial" (reduced to one equation) and not provides then enough information. This is typically due to the fact that we have used only a sub-set of the equations $\big($of bi-degree $(2^j, 1)\big)$. So, the open question is how we can use cleverly all the equations of $\mathsf{McE}_{k,n,r}(\mathbf{X}', \mathbf{Y}')$ in the binary case.

References

1. M. Baldi and G. F. Chiaraluce. Cryptanalysis of a new instance of McEliece cryptosystem based on QC-LDPC codes. In *IEEE International Symposium on Information Theory*, pages 2591–2595, Nice, France, March 2007.
2. Magali Bardet. *Étude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie*. PhD thesis, Université de Paris VI, 2004.
3. Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. Complexity study of Gröbner basis computation. Technical report, INRIA, 2002. http://www.inria.fr/rrrt/rr-5049.html.
4. Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations. In *Proc. International Conference on Polynomial System Solving (ICPSS)*, pages 71–75, 2004.
5. Magali Bardet, Jean-Charles Faugère, Bruno Salvy, and Bo-Yin Yang. Asymptotic behaviour of the degree of regularity of semi-regular polynomial systems. In *Proc. of MEGA 2005, Eighth International Symposium on Effective Methods in Algebraic Geometry*, 2005.
6. T. P. Berger, P.L. Cayrel, P. Gaborit, and A. Otmani. Reducing key length of the McEliece cryptosystem. In Bart Preneel, editor, *Progress in Cryptology - Second International Conference on Cryptology in Africa (AFRICACRYPT 2009)*, volume 5580 of *Lecture Notes in Computer Science*, pages 77–97, Gammarth, Tunisia, June 21-25 2009.
7. E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems. *Information Theory, IEEE Transactions on*, 24(3):384–386, May 1978.
8. E. R. Berlekamp. Factoring polynomials over finite fields. In E. R. Berlekamp, editor, *Algebraic Coding Theory*, chapter 6. McGraw-Hill, 1968.
9. D. J. Bernstein, T. Lange, and C. Peters. Attacking and defending the McEliece cryptosystem. In *PQCrypto*, volume 5299 of *LNCS*, pages 31–46, 2008.
10. B. Biswas and N. Sendrier. McEliece cryptosystem implementation: Theory and practice. In *PQCrypto*, volume 5299 of *LNCS*, pages 47–62, 2008.
11. Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, University of Innsbruck, 1965.
12. A. Canteaut and F. Chabaud. A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, 1998.
13. D. A. Cox, J. B. Little, and D. O'Shea. *Ideals, Varieties, and algorithms: an Introduction to Computational Algebraic Geometry and Commutative Algebra.* Undergraduate Texts in Mathematics, Springer-Verlag, New York., 2001.
14. J.-C. Faugère. A new efficient algorithm for computing gröbner bases (f4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, 1999.
15. J.-C. Faugère. A new efficient algorithm for computing gröbner bases without reduction to zero : F5. In *ISSAC'02*, pages 75–83. ACM press, 2002.
16. Jean-Charles Faugère, Mohab Safey El Din, and Pierre-Jean Spaenlehauer. Gröbner bases of bihomogeneous ideals generated by polynomials of bidegree (1,1): Algorithms and complexity. *CoRR*, abs/1001.4004, 2010.
17. Jean-Charles Faugère, Patrizia M. Gianni, Daniel Lazard, and Teo Mora. Efficient computation of zero-dimensional gröbner bases by change of ordering. *J. Symb. Comput.*, 16(4):329–344, 1993.
18. Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. Algebraic cryptanalysis of McEliece variants with compact keys. In *Proceedings of Eurocrypt 2010*. Springer Verlag, 2010. to appear.
19. P. Gaborit. Shorter keys for code based cryptography. In *Proceedings of the 2005 International Workshop on Coding and Cryptography (WCC 2005)*, pages 81–91, Bergen, Norway, March 2005.
20. P. J. Lee and E. F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In *Advances in Cryptology - EUROCRYPT'88*, volume 330/1988 of *Lecture Notes in Computer Science*, pages 275–280. Springer, 1988.
21. J. S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, 34(5):1354–1359, 1988.
22. P. Loidreau and N. Sendrier. Weak keys in the mceliece public-key cryptosystem. *IEEE Transactions on Information Theory*, 47(3):1207–1211, 2001.
23. F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes.* North–Holland, Amsterdam, fifth edition, 1986.

24. R. J. McEliece. *A Public-Key System Based on Algebraic Coding Theory*, pages 114–116. Jet Propulsion Lab, 1978. DSN Progress Report 44.
25. R. Misoczki and P. S. L. M. Barreto. Compact McEliece keys from Goppa codes. In *Selected Areas in Cryptography (SAC 2009)*, Calgary, Canada, August 13-14 2009.
26. A. Otmani, J.P. Tillich, and L. Dallot. Cryptanalysis of McEliece cryptosystem based on quasi-cyclic ldpc codes. In *Proceedings of First International Conference on Symbolic Computation and Cryptography*, pages 69–81, Beijing, China, April 28-30 2008. LMIB Beihang University.
27. N. Patterson. The algebraic decoding of Goppa codes. *IEEE Transactions on Information Theory*, 21(2):203–207, 1975.
28. N. Sendrier. Finding the permutation between equivalent linear codes: The support splitting algorithm. *IEEE Transactions on Information Theory*, 46(4):1193–1203, 2000.
29. J. Stern. A method for finding codewords of small weight. In G. D. Cohen and J. Wolfmann, editors, *Coding Theory and Applications*, volume 388 of *Lecture Notes in Computer Science*, pages 106–113. Springer, 1988.

A Hilbert Bi-Series

We say that an ideal is *bihomogeneous* if there exists a set of bihomogeneous generators. The vector space of bihomogeneous polynomials of bi-degree (α,β) in a polynomial ring R will be denoted by $R_{\alpha,\beta}$. If $\mathscr{I}$ is a bihomogeneous ideal, then $I_{\alpha,\beta}$ will denote the vector space $I \cap R_{\alpha,\beta}$.

Definition 2 ([16]). *Let $\mathscr{I}$ be a bihomogeneous ideal of R. The Hilbert bi-series is defined by*

$$\mathrm{HS}_{\mathscr{I}}(t_1,t_2) = \sum_{(\alpha,\beta)\in\mathbb{N}^2} \dim(R_{\alpha,\beta}/I_{\alpha,\beta})t_1^{\alpha}t_2^{\beta}.$$

For bi-regular bilinear systems, [16] provide an explicit form of the bi-series.

Theorem 2. *Let $f_1,\dots,f_m \in R$ be a bi-regular bilinear sequence, with $m \le n_{X'}+n_{Y'}$. Then*

$$\mathrm{HS}_{I_m}(t_1,t_2) = \frac{(1-t_1t_2)^m + N_{n_{X'}+1}(t_1,t_2) + N_{n_{Y'}+1}(t_1,t_2)}{(1-t_1)^{n_{X'}+1}(1-t_2)^{n_{Y'}+1}},$$

where

$$N_n(t_1,t_2) = t_1t_2(1-t_2)^n \sum_{\ell=1}^{m-n} (1-t_1t_2)^{m-n-\ell}\left[1-(1-t_1)^{\ell}\sum_{k=1}^{n} t_1^{n-k}\binom{\ell+n-k-1}{n-k}\right].$$

A variant of the F4 algorithm

Antoine Joux[1,2] and Vanessa Vitse[2]

[1] Direction Générale de l'Armement (DGA)
[2] Université de Versailles Saint-Quentin, Laboratoire PRISM, 45 av. des États-Unis, 78035 Versailles cedex, France
`antoine.joux@m4x.org` `vanessa.vitse@prism.uvsq.fr`

Abstract. Algebraic cryptanalysis usually requires to find solutions of several similar polynomial systems. A standard tool to solve this problem consists of computing the Gröbner bases of the corresponding ideals, and Faugère's F4 and F5 are two well-known algorithms for this task. In this paper, we adapt the "Gröbner trace" method of Traverso to the context of the F4 algorithm. The resulting variant of F4 is well suited to algebraic attacks of cryptosystems since it is designed to compute Gröbner bases of a set of polynomial systems having the same shape. It is faster than F4 as it avoids all reductions to zero, but preserves its simplicity and its computation efficiency, thus competing with F5.

Key words: Gröbner basis, Gröbner trace, F4, F5, multivariate cryptography, algebraic cryptanalysis

1 Introduction

The goal of algebraic cryptanalysis is to break cryptosystems by using mathematical tools coming from symbolic computation and modern algebra. More precisely, an algebraic attack can be decomposed in two steps: first the cryptosystem and its specifics have to be converted into a set of multivariate polynomial equations, then the solutions of the obtained polynomial system have to be computed. The security of a cryptographic primitive thus strongly relies on the difficulty of solving the associated polynomial system. These attacks have been proven to be very efficient for both public key or symmetric cryptosystems and stream ciphers (see [2] for a thorough introduction to the subject).

In this article, we focus on the polynomial system solving part. It is well known that this problem is very difficult (NP-hard in general). However, for many instances coming from algebraic attacks, the resolution is easier than in the worst-case scenario. Gröbner bases, first introduced in [6], are a fundamental tool for tackling this problem. Historically, one can distinguish two families of Gröbner basis computation algorithms: the first one consists of developments of Buchberger's original algorithm [8, 14, 15, 19], while the second can be traced back to the theory of elimination and resultants and relies on Gaussian elimination of Macaulay matrices [10, 24–26]. Which algorithm to use depends on the shape and properties of the cryptosystem and its underlying polynomial system (base field, degrees of the polynomials, number of variables, symmetries...).

Faugère's F4 algorithm [14] combines ideas from both families. It is probably the most efficient installation of Buchberger's original algorithm, and uses Gaussian elimination to speed up the time-consuming step of "critical pair" reductions. It set new records in Gröbner basis computation when it was published a decade ago, and its implementation in Magma [5] is still considered as a major reference today. However, F4 shares the main drawback of Buchberger's algorithm, namely, it spends a lot of time computing useless reductions. This issue was addressed by Faugère's next algorithm, F5 [15], which first rose to fame with the cryptanalysis of the HFE Challenge [16]. Since then, it has been successfully used to break several other cryptosystems (e.g. [4, 17]), increasing considerably the popularity of algebraic attacks. It is often considered as the most efficient algorithm for computing Gröbner bases over finite fields and its remarkable performances are for the main part attributable to the use of an elaborate criterion. Indeed, the F5 criterion allows to skip much more unnecessary critical pairs than the classical Buchberger's criteria [7]; actually it eliminates a priori all reductions to zero under the mild assumption that the system forms a semi-regular sequence [3]. Nevertheless, this comes at the price of degraded performances in the reduction step: the polynomials considered in the course of the F5 algorithm are “top-reduced”, but their tails are left almost unreduced because many reductions are forbidden for “signature” compatibility conditions.

In many instances of algebraic attacks, one has to compute Gröbner bases for numerous polynomial systems that have the same shape, and whose coefficients are either random or depend on a relatively small number of parameters. In this context, one should use specifically-devised algorithms that take this information into account. A first idea would be to compute a parametric or *comprehensive Gröbner basis* [30]; its specializations yield the Gröbner bases of all the ideals in a parametric polynomial system. However, for the instances arising in cryptanalysis, the computational cost of a comprehensive Gröbner basis is prohibitive. Another method was proposed by Traverso in the context of modular computations of rational Gröbner bases [29]: by storing the *trace* of an initial execution of the Buchberger algorithm, one can greatly increase the speed of almost all subsequent computations. Surprisingly, it seems that this approach was never applied to cryptanalysis until now.

We present in this paper how a similar method allows to avoid all reductions to zero in the F4 algorithm after an initial precomputation. A list of relevant critical pairs is extracted from a first F4 execution, and is used for all following computations; the precomputation overhead is largely compensated by the efficiency of the F4 reduction step, yielding theoretically better performances than F5. This algorithm is by nature probabilistic: the precomputed list is in general not valid for all the subsequent systems. One of the main contribution of this article is to give a complete analysis of this F4 variant and to estimate its probability of failure, which is usually very small.

The paper is organized as follows. After recalling the basic structure of Buchberger-type algorithms, we explain in section 2 how to adapt it to the context of several systems of the same shape. We then give detailed pseudo-code of our variant of F4, which consists of the two routines `F4Precomp` and `F4Remake`, for the first precomputation and the subsequent iterations respectively. In section 3, we recall the mathematical frame for the otherwise imprecise notion of “similar looking systems” and derive probability estimates for the correctness of our algorithm, depending on the type of the system and the size of the base field. We also compare the complexities of our variant and of F5, and explain when it is better to use our algorithm. The last section is devoted to applications: the first example is the index calculus method of [20] and is a typical case where our algorithm outperforms F4 and F5. We then show how it fits into the hybrid approach of [4] and consider the example of the cryptanalysis of the UOV signature scheme [22]. The next example is provided by the Kipnis-Shamir attack on the MinRank problem: we compare our results to those of [17]. Finally, we evaluate the performances of our F4 variant on the classical `Katsura` benchmarks.

We would like to mention that the Gröbner trace method has already been applied to Faugère’s algorithms for the decoding of binary cyclic codes [1]; however, the analysis therein was limited to this very specific case, and no implementation details nor probability estimates were given. This idea was then independently rediscovered in our previous article [20], where it was applied to the discrete log problem on elliptic curves.

2 The F4 variant

2.1 Description of the algorithm

We begin by recalling the standard characterization of Gröbner bases:

Theorem 1 ([8]) *A family $G = \{g_1, \dots, g_s\}$ in $\mathbb{K}[X_1, \dots, X_n]$ is a Gröbner basis if and only if for all $1 \le i < j \le s$, the remainder of $S(g_i, g_j)$ on division by G is zero, where $S(g_i, g_j)$ is the S-polynomial of g_i and g_j: $S(g_i, g_j) = \dfrac{LM(g_i) \vee LM(g_j)}{LT(g_i)} g_i - \dfrac{LM(g_i) \vee LM(g_j)}{LT(g_j)} g_j$.*

It is straightforward to adapt this result into the Buchberger's algorithm [8], which outputs a Gröbner basis of an ideal $I = \langle f_1, \dots, f_r \rangle$: one computes iteratively the remainder by G of every possible S-polynomials and appends this remainder to G whenever it is different from zero. In the following, we will rather work with critical pairs instead of S-polynomials: the critical pair of two polynomials f_1 and f_2 is defined as the tuple $(lcm, u_1, f_1, u_2, f_2)$ where $lcm = LM(f_1) \vee LM(f_2)$ and $u_i = \frac{lcm}{LM(f_i)}$.

The reduction of critical pairs is by far the biggest time-consuming part of the Buchberger's algorithm. The main idea of Faugère's F4 algorithm is to use linear algebra to simultaneously reduce a large number of pairs. At each iteration step, a Macaulay-style matrix is constructed, whose columns correspond to monomials and rows to polynomials. This matrix contains the products $(u_i f_i)$ coming from the selected critical pairs (classically, all pairs with the lowest total degree lcm, but other selection strategies are possible) and also all polynomials involved in their reductions, which are determined during the preprocessing phase. By computing the reduced row echelon form of this matrix, we obtain the reduced S-polynomials of all pairs considered. This algorithm, combined with an efficient implementation of linear algebra, yields very good results.

As mentioned in the introduction, F4 has the drawback of computing many useless reductions to zero, even when the classical criteria of Buchberger [7] are taken into account. But when one has to compute several Gröbner bases of similar polynomial systems, it is possible to avoid, in most cases, all reductions to zero by means of a precomputation on the first system. Here is the outline of our F4 variant:

1. For precomputation purposes, run a standard F4 algorithm on the first system, with the following modifications:
   – At each iteration, store the list of all polynomial multiples (u_i, f_i) coming from the critical pairs.
   – During the row echelon computing phase, reductions to zero correspond to linear dependency relations between the rows of the matrix; for each such relation, remove a multiple (u_i, f_i) from the stored list.
2. For each subsequent system, run a F4 computation with the following modifications:
   – Do not maintain nor update a queue of untreated pairs.
   – At each iteration, instead of selecting pairs from the queue, pick directly from the previously stored list all the relevant multiples (u_i, f_i).

2.2 Pseudo-code

We now give the detailed pseudo-code of the `F4Precomp` algorithm which performs the precomputation, and of the `F4Remake` algorithm which is used for the subsequent systems.

The precomputation

Given a family of polynomials $\{f_1, \dots, f_r\}$, the `F4Precomp` algorithm computes for each iteration step of the classical F4 algorithm, the list of polynomial multiples that will be used by `F4Remake` on subsequent computations. This algorithm follows very closely [14], with the following additional features:

– A list L of lists of couples is introduced; at the end of the i-th main iteration, $L[i]$ contains the desired list of polynomial multiples for that step. Each polynomial multiple is represented by a couple (m, n), where m is a monomial and n is the index of the polynomial in a global list G (this list G will be progressively reconstructed by `F4Remake`). In the same way, a list L_{tmp} is used to temporary store these couples.
– Instead of just computing the reduced row echelon form M' of the matrix M, we also compute an auxiliary matrix A such that $AM = M'$. If reductions to zero occur, then the bottom part of M' is null and the corresponding bottom part of A gives the linear dependencies between the rows of M. This information is exploited in lines 23 to 29, in order to remove from the temporary list L_{tmp} the useless multiples before copy in $L[step]$. Actually, only the bottom-left part A' of A is of interest: it contains the linear dependencies between the rows of M coming from the critical pairs, modulo those coming from the preprocessing. It is clear that with each dependency relation, one polynomial multiple can be removed, but some care must be taken in this choice. To do so, the row echelon form $\tilde{A}$ of A' is then computed and

the polynomial multiples corresponding to the pivots of $\tilde{A}$ are removed. Among the remaining polynomial multiples, those whose leading monomial is now unique can also be removed.

Apart from these modifications, the pseudo-code is basically the F4 algorithm with Gebauer and Möller installation of the BuchBerger's criteria (`Update` subroutine) [19]. The only notable change concerns the implementation of the `Simplify` procedure: instead of searching through all the former matrices and their row echelon forms for the adequate simplification as in [14], we introduce an array $TabSimplify$ which contains for each polynomial f in the basis a list of couple of the form $(m, g) \in T \times \mathbb{K}[\underline{X}]$, meaning that the product mf can be simplified into the more reduced polynomial g. This array is updated after the reduced row echelon form is computed (lines 13 to 18 of `Postprocessing`).

Alg. 1 F4Precomp

```
INPUT :  $f_1, \dots, f_r \in \mathbb{K}[\underline{X}]$
OUTPUT :  a list of lists of couples $(m, n) \in T \times \mathbb{N}$
 1. $G \leftarrow [\,]$, $G_{min} \leftarrow \emptyset$, $P \leftarrow \emptyset, TabSimplify \leftarrow [\,], L \leftarrow [\,]$
 2. for $i = 1$ to $r$ do
 3.     $G[i] \leftarrow f_i$
 4.     $TabSimplify[i] \leftarrow [(1, f_i)]$
 5.     $Update(f_i)$
 6. $step = 1$
 7. while $P \neq \emptyset$ do
 8.     $P_{sel} \leftarrow Sel(P)$
 9.     $F \leftarrow [\,]$, $LM(F) \leftarrow \emptyset$, $T(F) \leftarrow \emptyset$, $L[step] \leftarrow [\,]$, $L_{tmp} \leftarrow [\,]$
10.     for all $pair = (lcm, t_1, g_1, t_2, g_2) \in P_{sel}$ do
11.         for $k = 1$ to $2$ do
12.             $ind \leftarrow index(g_k, G)$
13.             if $(t_k, ind) \notin L_{tmp}$ then
14.                 $Append(L_{tmp}, (t_k, ind))$
15.                 $f \leftarrow Simplify(t_k, ind)$
16.                 $Append(F, f)$
17.                 $LM(F) \leftarrow LM(F) \cup \{LM(f)\}$
18.                 $T(F) \leftarrow T(F) \cup \{m \in T : m$ monomial of $f\}$
19.     $Preprocessing(F, T(F), LM(F))$
20.     $M \leftarrow$ matrix whose rows are the polynomials in $F$
21.     $(M'|A) \leftarrow ReducedRowEchelonForm(M|I_{\#F})$ $(\Rightarrow AM = M')$
22.     $rank \leftarrow Postprocessing(M', LM(F))$
23.     if $rank < \#F$ then
24.         $A' \leftarrow A[rank + 1..\#F][1..\#L_{tmp}]$
25.         $\tilde{A} \leftarrow ReducedRowEchelonForm(A')$
26.         $C \leftarrow \{c \in \{1, \dots, \#L_{tmp}\} : c$ is not a column number of a pivot in $\tilde{A}\}$
27.         for $j \in C$ do
28.             if $\exists k \in C$ such that $k \neq j$ and $LM(F[k]) = LM(F[j])$ then
29.                 $Append(L[step], L_{tmp}[j])$
30.     $step \leftarrow step + 1$
31. return $L$
```

In the pseudo-code, some variables are supposed to be global: G, a list of polynomials that forms a basis of $\langle f_1, \dots, f_r \rangle$; G_{min}, a set of polynomials which is the minimalized version of G; $TabSimplify$, an array of lists of couples used for the simplification of polynomials multiples; P, a queue of yet untreated critical pairs. The function Sel on line 8 is a selection function, whose expression depends on the chosen strategy; usually, selecting all pairs of lowest total degree lcm (normal strategy) yields the best performances. The notation $index(g, G)$ stands for the integer i such that $G[i] = g$, and the function $pair(f_1, f_2)$ outputs the critical pair $(lcm, u_1, f_1, u_2, f_2)$. Finally, $ReducedRowEchelonForm$ computes as expected the reduced row echelon form of its input matrix. We stress that great care should be taken in the implementation of this last function since almost all the execution time of the algorithm is spent in it. Note that the test on line 15 in `Update` is only necessary during the initialisation phase of `F4Precomp` (line 5).

Alg. 2 Update

```
INPUT : $f \in \mathbb{K}[\underline{X}]$
 1. for all $pair = (lcm, t_1, g_1, t_2, g_2) \in P$ do
 2.   if $(LM(f) \vee LM(g_1)$ divides strictly $lcm)$ AND $(LM(f) \vee LM(g_2)$ divides strictly $lcm)$ then
 3.     $P \leftarrow P \setminus \{pair\}$
 4. $P_0 \leftarrow \emptyset, P_1 \leftarrow \emptyset, P_2 \leftarrow \emptyset$
 5. for all $g \in G_{min}$ do
 6.   if $LM(f) \wedge LM(g) = 1$ then
 7.     $P_0 \leftarrow P_0 \cup pair(f, g)$
 8.   else
 9.     $P_1 \leftarrow P_1 \cup pair(f, g)$
10. for all $pair = (lcm, t_1, g_1, t_2, g_2) \in P_1$ do
11.   $P_1 \leftarrow P_1 \setminus \{pair\}$
12.   if $\nexists pair' = (lcm', t_1', g_1', t_2', g_2') \in P_0 \cup P_1 \cup P_2$ such that $lcm'|lcm$ then
13.     $P_2 \leftarrow P_2 \cup \{pair\}$
14. $P \leftarrow P \cup P_2$
15. if $\nexists g \in G_{min}$ such that $LM(g)|LM(f)$ then
16.   for all $g \in G_{min}$ do
17.     if $LM(f)|LM(g)$ then
18.       $G_{min} \leftarrow G_{min} \setminus \{g\}$
19.     $G_{min} \leftarrow G_{min} \cup \{f\}$
```

Alg. 3 Preprocessing

```
INPUT : $F, T(F), LM(F)$
 1. $Done \leftarrow LM(F)$
 2. while $T(F) \neq Done$ do
 3.   $m \leftarrow \max(T(F) \setminus Done)$
 4.   $Done \leftarrow Done \cup \{m\}$
 5.   for all $g \in G_{min}$ do
 6.     if $LM(g)|m$ then
 7.       $g' \leftarrow Simplify\left(\frac{m}{LM(g)}, index(g, G)\right)$
 8.       $Append(F, g')$
 9.       $LM(F) \leftarrow LM(F) \cup \{m\}$
10.       $T(F) \leftarrow T(F) \cup \{m' \in T : m'$ monomial of $g'\}$
11.       break
```

Alg. 4 Simplify

```
INPUT : $t \in T, ind \in \mathbb{N}$
OUTPUT : $p \in \mathbb{K}[\underline{X}]$
 1. for $(m, f) \in TabSimplify[ind]$ (from last to first) do
 2.   if $m = t$ then
 3.     return $f$
 4.   else
 5.     if $m|t$ then
 6.       $Append\left(TabSimplify[ind], \left(m, \frac{t}{m}f\right)\right)$
 7.       return $\frac{t}{m}f$
```

Alg. 5 Postprocessing

```
INPUT :  a matrix $M$ in reduced row echelon form with $\#F$ lines and an ordered set of monomials $LM(F)$
OUTPUT :  the rank of the matrix $M$
 1. for $i = 1$ to $\#F$ do
 2.     $f \leftarrow M[i]$
 3.     if $f = 0$ then
 4.         break
 5.     if $LM(f) \notin LM(F)$ then
 6.         $Append(G, f)$
 7.         $Update(f)$
 8.         $TabSimplify[\#G] \leftarrow [(1, f)]$
 9.     else
10.         for $g \in G_{min}$ do
11.             $ind \leftarrow index(g, G)$
12.             if $LM(g)|LM(f)$ then
13.                 for $j = 1$ to $\#TabSimplify[ind]$ do
14.                     if $TabSimplify[ind][j] = \left(\frac{LM(f)}{LM(g)}, .\right)$ then
15.                         $TabSimplify[ind][j] = \left(\frac{LM(f)}{LM(g)}, f\right)$
16.                         break
17.                 if $j > \#TabSimplify[ind]$ then
18.                     $Append\left(TabSimplify[ind], \left(\frac{LM(f)}{LM(g)}, f\right)\right)$
19. return  $i - 1$
```

F4Remake

The `F4Remake` algorithm uses the same routines `Simplify`, `Preprocessing` and `Postprocessing`. Since it no longer uses critical pairs, the subroutine `Update` can be greatly simplified and is replaced by `Update2`.

Alg. 6 F4Remake

```
INPUT :  $f_1, \dots, f_r \in \mathbb{K}[\underline{X}]$, a list $L$ of lists of couples $(m, n) \in T \times \mathbb{N}$
OUTPUT :  $G_{min}$, the reduced minimal Gröbner basis of $f_1, \dots, f_r$
 1. $G \leftarrow [\,],\ G_{min} \leftarrow \emptyset, TabSimplify \leftarrow [\,]$
 2. for $i = 1$ to $r$ do
 3.     $G[i] \leftarrow f_i$
 4.     $TabSimplify[i] \leftarrow [(1, f_i)]$
 5.     $Update2(f_i)$
 6. for $step = 1$ to $\#L$ do
 7.     $F \leftarrow [\,]$, $LM(F) \leftarrow \emptyset$, $T(F) \leftarrow \emptyset$
 8.     for all $(m, n) \in L[step]$ do
 9.         if $n > \#G$ then
10.             computation fails ! exit
11.         $f \leftarrow Simplify(m, n)$, $Append(F, f)$
12.         $LM(F) \leftarrow LM(F) \cup \{LM(f)\}$
13.         $T(F) \leftarrow T(F) \cup \{m \in T : m$ monomial of $f\}$
14.     $Preprocessing(F, T(F), LM(F))$
15.     $M \leftarrow$ matrix whose rows are the polynomials in $F$
16.     $M' \leftarrow ReducedRowEchelonForm(M)$
17.     $Postprocessing(M', LM(F))$
18. return  $InterReduce(G_{min})$
```

```
Alg. 7 Update2
INPUT : $f \in \mathbb{K}[\underline{X}]$
 1. if $\nexists g \in G_{min}$ such that $LM(g)|LM(f)$ then
 2.     for all $g \in G_{min}$ do
 3.         if $LM(f)|LM(g)$ then
 4.             $G_{min} \leftarrow G_{min} \setminus \{g\}$
 5.         $G_{min} \leftarrow G_{min} \cup \{f\}$
```

2.3 Further developments

The above pseudo-code of `F4Remake` does not check the correctness of the computation, except for the basic verification of line 10. More tests could be easily included: for instance, it is possible to store during the precomputation the leading monomials of the generators created at each step, and check in `F4Remake` if the new generators have the correct LM. In case of a failed computation, proper error handling would be recommended, e.g. either by restarting or resuming the computation with the standard F4. At the end of the execution, a last check would be to verify whether the result (which is always a basis of the ideal) is indeed a Gröbner basis. This can be quite expensive, but is usually unnecessary: indeed, the output is always correct if the sets of leading monomials of the bases returned by `F4Remake` and `F4Precomp` coincide, assuming that the precomputation behaved generically (see section 3). Anyway, when the ideal is zero-dimensional with a small degree (as is often the case in the context of algebraic attacks), a verification is almost immediate.

It is also possible to store during precomputation all the relevant polynomial multiples appearing in the matrices M, instead of only those arising from the critical pairs. This increases considerably the size of `F4Precomp`'s output, but allows to skip the preprocessing phase in `F4Remake`. However, the gain provided by this optimization is relatively minor, since the cost of the preprocessing is usually small compared to the computation of the reduced row echelon form.

A different approach is outlined in [1]: instead of recording the information about relevant polynomials in a file, the precomputation directly outputs a program (in the C language) containing the instructions for the subsequent computations. Clearly, this code generating technique is much more complicated, but should be faster even when the compilation time of the output program is taken into account.

3 Analysis of the algorithm and complexity

3.1 Similar systems

Our algorithm is designed to be applied on many systems of the "same shape". If $\{f_1, \ldots, f_r\}$ and $\{f'_1, \ldots, f'_r\}$ are two similarly-looking polynomial systems, we want to estimate the probability that our algorithm computes the Gröbner basis of the second system, the precomputation having been done with the first system. This requires some more precise definitions.

Definition 2 *A generic polynomial F of degree d in n variables $X_1, \ldots, X_n$ is a polynomial with coefficients in $\mathbb{K}[\{Y_{i_1,\ldots,i_n}\}_{i_1+\ldots+i_n \le d}]$ of the form $F = \sum_{i_1+\ldots+i_n \le d} Y_{i_1,\ldots,i_n} X_1^{i_1} \ldots X_n^{i_n}$.*

A generic polynomial is thus a polynomial in which each coefficient is a distinct variable. Such polynomials are interesting to study because a system of random polynomials $f_1, \ldots, f_r$ (i.e. such that each coefficient is random) of total degree $d_1, \dots, d_r$ respectively, is expected to behave like the corresponding system of generic polynomials.

Let $F_1, \dots, F_r$ be a system of generic polynomials. If we consider F_i as an element of $\mathbb{K}(\underline{Y})[\underline{X}]$, we can compute the Gröbner basis of this system with the F4 algorithm, at least theoretically (in practice, the rational fraction coefficients will likely become extremely large). Now let $f_1, \dots, f_r$ be a random system with $\deg(f_i) = \deg(F_i)$. We say that $f_1, \dots, f_r$ behaves generically if we encounter the same number of iterations as with $F_1, \dots, F_r$ during the computation of its Gröbner basis using F4, and if the same number of new polynomials with the same leading monomials are generated at each step of the algorithm. We will now translate this condition algebraically. Assume that the system $f_1, \dots, f_r$ behaves generically until the $(i-1)$-th step; this implies in particular that the critical pairs involved at step i for both systems are similar, in the following sense: $(lcm, u_1, p_1, u_2, p_2)$ is similar to $(lcm', u_1', p_1', u_2', p_2')$ if $LM(p_1) = LM(p_1')$ and $LM(p_2) = LM(p_2')$ (so that $u_i = u_i'$ and $lcm = lcm'$).

Let M_g be the matrix of polynomial multiples constructed by F4 at step i for the generic system, and M be the one for $f_1, \dots, f_r$. It is possible that after the preprocessing M is smaller than M_g, but for the purpose of our discussion, we may assume that the missing polynomial multiples are added to M; the corresponding rows will have no effect whatsoever later in the algorithm. Thus the k-th rows of M and M_g, seen as polynomials, have identical leading monomial; we note s the number of distinct leading monomials in M (or M_g). If we compute the reduced row echelon form of M_g, up to a well-chosen permutation of columns we obtain

$$\tilde{M_g} = \left(\begin{array}{c|c} I_{rank} & A \\ \hline 0 & 0 \end{array}\right)$$

Using the same transformations on M with adapted coefficients, we obtain a matrix

$$\tilde{M} = \left(\begin{array}{cc|c} I_s & C & \\ \cline{1-2} 0 & B & A' \\ \hline \multicolumn{2}{c|}{0} & 0 \end{array}\right)$$

where B is a square matrix of size $rank - s$. Then the system $f_1, \dots, f_r$ behaves generically at step i if and only if this matrix B is invertible. Finally, we obtain that the system behaves generically during the course of the F4 algorithm if at each step, the corresponding matrix B is invertible.

Heuristically, since the system is random, we will assume that these matrices B are random. This hypothesis will allow us to give estimates for the probability that a system behaves generically, using the following easy lemma:

Lemma 3 *Let $M = (m_{ij}) \in \mathcal{M}_n(\mathbb{F}_q)$ be a random square matrix, i.e. such that the coefficients m_{ij} are chosen randomly, independently and uniformly in $\mathbb{F}_q$. Then M is invertible with probability $\prod_{i=1}^{n}(1 - q^{-i})$. This probability is greater than the limit $c(q) = \prod_{i=1}^{\infty}(1 - q^{-i})$.*

When q is large, $c(q)$ is very close to $1 - 1/q$ and has the explicit lower bound $c(q) \geq \left(\frac{q-1}{q}\right)^{\frac{q}{q-1}}$.

Since a system behaves generically if and only if all the matrices B are invertible, we obtain the probability that our F4 variant works successfully:

Theorem 4 *The algorithm* `F4Remake` *outputs a Gröbner basis of a random system $f_1, \dots, f_r \in \mathbb{F}_q[\underline{X}]$ with a probability that is heuristically greater than $c(q)^{n_{step}}$, assuming that the precomputation has been done with* `F4Precomp` *in n_{step} steps, for a system $f_1^0, \dots, f_r^0 \in \mathbb{F}_q[\underline{X}]$ that behaves generically.*

For a system of generic polynomials, it is known that the number of steps n_{step} during the execution of F4 (for a degree-graded monomial order) is at most equal to the degree of regularity d_{reg} of the homogenized system, which is smaller than the Macaulay bound $\sum_{i=1}^{r}(\deg F_i - 1) + 1$ [24]; this bound is sharp when the system is underdetermined. Since $c(q)$ converges to 1 when q goes to infinity, for a fixed degree of regularity the probability of success of our algorithm will be very close to 1 when the base field $\mathbb{F}_q$ is sufficiently large.

In practice, it is rather uncommon to deal with completely random polynomials. For many applications, the involved polynomial systems actually depend on a small number of random parameters, hence a more general framework would be the following:

Definition 5 *Let $F_1, \dots, F_r$ be polynomials in $\mathbb{K}[Y_1, \dots, Y_\ell][\underline{X}]$. We call the image of the map*

$$\mathbb{K}^\ell \to \mathbb{K}[\underline{X}]^r, \quad y = (y_1, \dots, y_\ell) \mapsto (F_1(y), \dots, F_r(y))$$

a parametric family (or family for short) of systems. We call the system $(F_1, \dots, F_r)$ the generic parametric system of the family.

A system of generic polynomials is of course a special case of a generic parametric system. As above, the `F4Remake` algorithm will give correct results for systems $f_1, \dots, f_r$ in a family that behave like its associated generic parametric system. The probability that this happens is difficult to estimate since it obviously depends on the family considered, but is usually better than for systems of generic polynomials. An important class of examples is when the highest degree homogeneous part of the F_i has coefficients in $\mathbb{K}$ (instead of $\mathbb{K}[Y_1, \dots, Y_\ell]$). Then all systems of this parametric family behave generically until the first fall of degree occurs. As a consequence, the probability of success of our algorithm can be quite good even when the base field is relatively small, see section 4.2 for an example.

3.2 Change of characteristic

Another application of our algorithm is the computation of Gröbner bases of "random" polynomial systems over a large field, using a precomputation done over a small finite field. Even for a single system $f_1, \dots, f_r$ in $\mathbb{F}_p[\underline{X}]$, it is sometimes more advantageous to precompute the Gröbner basis of a system $f'_1, \dots, f'_r$ with $\deg f_i = \deg f'_i$ in $\mathbb{F}_{p'}[\underline{X}]$ for a small prime p', and then use `F4Remake` on the initial system, than to directly compute the Gröbner basis with F4. The estimated probabilities derived in section 3.1 do not directly apply to this situation, but a similar analysis can be done.

We recall that for every prime number p, there exists a well-defined reduction map $\mathbb{Q}[\underline{X}] \to \mathbb{F}_p[\underline{X}]$, which sends a polynomial P to $\bar{P} = cP \mod p$, where $c \in \mathbb{Q}$ is such that cP belongs to $\mathbb{Z}[\underline{X}]$ and is primitive (i.e. the gcd of its coefficients is one). Let $I = \langle f_1, \dots, f_r \rangle$ be an ideal of $\mathbb{Q}[\underline{X}]$, and let $\bar{I} = \langle \bar{f}_1, \dots, \bar{f}_r \rangle$ be the corresponding ideal in $\mathbb{F}_p[\underline{X}]$; we note $\{g_1, \dots, g_s\}$ the minimal reduced Gröbner basis of I. According to [12], we say that p is a "lucky" prime if $\{\bar{g}_1, \dots, \bar{g}_s\}$ is the minimal reduced Gröbner basis of $\bar{I}$, and "unlucky" otherwise. There is a weaker, more useful notion (adapted from [27]) of "F4 unlucky prime" or "weak unlucky prime": a prime number p is called so if the computation of the Gröbner bases of I and $\bar{I}$ with F4 differs. By doing the same analysis as in section 3.1, we can show that p is weakly unlucky if and only if one of the above-defined matrices B is not invertible. As before, these matrices can heuristically be considered as random and thus we obtain that the probability that a prime p is not weakly unlucky, is bounded from below by $c(p)^{n_{step}}$. So, if we want to compute the Gröbner basis of a system $f_1, \dots, f_r \in \mathbb{F}_p[\underline{X}]$ where p is a large prime, we can lift this system to $\mathbb{Q}[\underline{X}]$ and then reduce it to $f'_1, \dots, f'_r \in \mathbb{F}_{p'}[\underline{X}]$ where p' is a small prime number. Then we execute `F4Precomp` on the latter system and use the precomputation on the initial system with `F4Remake`. This will produce the correct result if p and p' are not weakly unlucky, thus p', while small enough so that the precomputation takes the least time possible, must be large enough so that the probability $c(p')^{n_{step}}$ is sufficiently close to 1.

In practice, this last approach should be used whenever possible. If one has to compute several Gröbner bases over a large field $\mathbb{F}_q$ of systems of the same parametric family, the precomputation should not be done over $\mathbb{F}_q$, but rather over a smaller field. We will adopt this strategy in almost all the applications presented in section 4.

3.3 Precomputation correctness

The output of `F4Precomp` is correct if the first system behaves generically; we have seen that this occurs with a good probability $c(q)^{n_{step}}$. We will now consider what can happen when the precomputation is not correct, and how to detect it. We can, at least theoretically, run `F4Remake` on the generic system; following Traverso's analysis [29] two cases are then possible:

1. This would produce an error. Then `F4Remake` will fail for most subsequent systems, so this situation can be easily detected after very few executions (the probability of no detection is very low: rough estimates have been given in [29] for the different characteristic case). More precisely, as soon as an error occurs with `F4Remake`, one has to determine whether the precomputation was incorrect or the current system does not behave generically. This can be done by looking at the course of the algorithm: if at some step `F4Remake` computes more new generators than `F4Precomp`, or generators with higher leading monomials, then we know at once that it is the precomputation which is incorrect.
2. The computation would succeed but the resulting output is not a Gröbner basis. This situation, while unlikely, is more difficult to detect: one has to check that the outputs of `F4Remake` on the first executions are indeed Gröbner bases. If there is a system for which this is not true, then the precomputation is incorrect.

Alternatively, one can run `F4Precomp` on several systems and check that the outputs coincide. If it is not the case, one should obviously select the most common output; the probability that a majority of precomputations is similarly incorrect is extremely low. Of course, if $c(q)^{n_{step}}$ is sufficiently close to 1, then the probability of an incorrect precomputation is low enough not to have to worry about these considerations.

3.4 Complexity

Generally, it is difficult to obtain good estimates for the complexity of Gröbner basis computation algorithms, especially of those based on Buchberger's approach. However, we can give a broad upper bound of the complexity of `F4Remake`, by observing that it can be reduced to the computation of the row echelon form of a D-Macaulay matrix of the homogenized system, whose useless rows would have been removed. In the case of generic systems, D is equal to the degree of regularity d_{reg} of the homogenized system. Thus we have an upper-bound for the complexity of our algorithm:

Proposition 6 *The number of field operations performed by* `F4Remake` *on a system of random polynomials over* $\mathbb{K}[X_1, \ldots, X_n]$ *is bounded by*

$$O\left(\binom{d_{reg}+n}{n}^{\omega}\right)$$

where d_{reg} *is the degree of regularity of the homogenized system and* ω *is the constant of matrix multiplication.*

Since there is no reduction to zero as well with F5 (under the assumption that the system is semi-regular), the same reasoning applies and gives the same upper-bound, cf [3]. However, we emphasize that these estimates are not really sharp and do not reflect the difference in performances between the two algorithms. Indeed, `F4Remake` has two main advantages over F5: the polynomials it generates are fully reduced, and it avoids the incremental structure of F5. More precisely, the F5 criterion relies on the use of a signature or label for each polynomial, and we have already mentioned in the introduction that signature compatibility conditions prohibit some reductions; therefore, the polynomials generated by F5 are not completely reduced, or are even redundant [13]. This incurs either more costly reductions later in the algorithm or a larger number of critical pairs. Secondly, the incremental nature of F5 implies that the information provided by the last system polynomials cannot be used to speed up the first stages of the computation.

Thus, our F4 variant should be used preferentially as soon as several Gröbner bases have to be computed and the base field is large enough for this family of systems. Nevertheless, the F5 algorithm remains irreplaceable when the Gröbner basis of only one system has to be computed, when the base field is too small (in particular over $\mathbb{F}_2$) or when the systems are so large that a precomputation would not be realisable.

4 Applications

In all applications, the variant `F4Remake` is compared with an implementation of F4 which uses the same primitives and structures (in language C), and also with the proprietary software Magma (V2.15-15) whose implementation is probably the best publicly available for the considered finite fields. Unless otherwise specified, all tests are performed on a 2.6 GHz Intel Core 2 Duo processor and times are given in seconds.

4.1 Index calculus

An index calculus method has been recently proposed in [11, 18] for the resolution of discrete logarithm on $E(\mathbb{F}_{q^n})$ where E is an elliptic curve defined over a small degree extension field. In order to find "relations", they make use of Semaev's idea [28] which allows to convert the relation search into the resolution of a multivariate polynomial system. A variation of this approach is given in [20], where relations with a slightly different form are considered: it has the advantage of leading to overdetermined systems and is thus faster in practical cases. We focus on the resolution of the polynomial systems arising from this last attack in the special case of $E(\mathbb{F}_{p^5})$ where p is a prime number. The polynomial systems in this example fit into the framework of parametric families: the coefficients polynomially depend on the x-coordinate of a random point $R \in E(\mathbb{F}_{p^5})$ (and also of the equation of the curve E). Our algorithm is particularly relevant for this example because of the large number of relations to collect, leading to an average of $4!p^2$ systems to solve. Moreover, p is large in all applications so the probability of success of our F4 variant is extremely good.

We cite directly the results from [20], where the `F4Remake` algorithm has first been introduced. The systems to solve are composed of 5 equations defined over $\mathbb{F}_p$ of total degree 8 in 4 variables. Degrevlex Gröbner bases of the corresponding ideals over several prime fields of size 8, 16, 25 and 32 bits are computed. The probabilities of failure are estimated under the assumption that the systems are random, and knowing that the computation takes 29 steps.

size of p	est. failure probability	F4Precomp	F4Remake	F4	F4 Magma
8 bits	0.11	8.963	2.844	5.903	9.660
16 bits	4.4×10^{-4}	(19.07)	3.990	9.758	9.870
25 bits	2.4×10^{-6}	(32.98)	4.942	16.77	118.8
32 bits	5.8×10^{-9}	(44.33)	8.444	24.56	1046

Fig. 1. Experimental results on $E(\mathbb{F}_{p^5})$

As explained in section 3.2, it is sufficient to execute the precomputation on the smaller field to get a list of polynomial multiples that works for the other cases; the timings of `F4Precomp` over the fields of size $16, 25$ and 32 bits are thus just indicative. The above figures show that the precomputation overhead is largely compensated as soon as there are more than two subsequent computations. Note that it would have been hazardous to execute `F4Precomp` on a smaller field as the probability of failure increases rapidly. It is mentioned in [20] that the systems have also been solved with a personal implementation of F5, and that the size of the Gröbner basis it computes at the last step before minimalization is surprisingly large (17249 labeled polynomials against no more than 2789 polynomials for both versions of F4). As a consequence, the timings of F5 obtained for these systems are much worse than those of F4 or its variants. This shows clearly that on this example, it is much more efficient to apply our algorithm rather than F5.

4.2 Hybrid approach

The hybrid approach proposed in [4] relies on a trade-off between exhaustive search and Gröbner basis computation. The basic idea is that when one wants to find a solution of a given system $f_1, \dots, f_r \in \mathbb{K}[X_1, \dots, X_n]$, it is sometimes faster to try to guess a small number of variables $X_1, \dots, X_k$. For each possible k-tuple $(x_1, \dots, x_k)$, one computes the Gröbner basis of the corresponding specialized system $f_1(x_1, \dots, x_k), \dots, f_r(x_1, \dots, x_k) \in \mathbb{K}[X_{k+1}, \dots, X_n]$ until a solution is found; the advantage is that the specialized systems are much simpler to solve than the initial one.

The hybrid approach is thus a typical case when many systems of the same shape have to be solved and fits perfectly into the framework of parametric families we have described in section 3.1. However, this method is most useful when the search space is reasonably small, which implies in particular that the size of the base field cannot be too large, so one should be wary of the probability of success before applying our F4 variant to this context.

As an example, we consider the cryptanalysis of the Unbalanced Oil and Vinegar system (UOV, [22]), described in [4]. Briefly, the attack can be reduced to the resolution of a system of n quadratic equations in n variables over a finite field $\mathbb{K}$; for the recommended set of parameters, $n = 16$ and $\mathbb{K} = \mathbb{F}_{16}$. Although the base field is quite small, our F4 variant has rather good results in this cryptanalysis: this is due to the fact that the quadratic part of the evaluated polynomials $f_i(x_1, \dots, x_k) \in \mathbb{K}[X_{k+1}, \dots, X_n]$ does not depend on the values of the specialized variables $X_1, \dots, X_k$, and hence all the systems behave generically until the first fall of degree. For instance, for $k = 3$ the computation with F4 takes 6 steps, and no fall of degree occurs before the penultimate step, so a heuristic estimation of the probability of success is $c(16)^2 \simeq 0.87$. To check this estimate we have performed an exhaustive exploration of the search space $\mathbb{F}_{16}^3$ using `F4Remake`. The actual probability of success is 80.859%, which is satisfying but somewhat smaller than estimated. The difference can be readily explained by the fact that the systems are not completely random.

4.3 MinRank

We briefly recall the MinRank problem: given $m + 1$ matrices $M_0, M_1, \dots, M_m \in \mathcal{M}_n(\mathbb{K})$ and a positive integer r, is there a m-tuple $(\alpha_1, \dots, \alpha_m) \in \mathbb{K}^m$ such that $\text{Rank}\left(\sum_{i=1}^{m} \alpha_i M_i - M_0\right) \leq r$.

We focus on the challenge A proposed in [9]: $\mathbb{K} = \mathbb{F}_{65521}; m = 10; n = 6; r = 3$. The Kipnis-Shamir's attack converts instances of the MinRank problem into quadratic multivariate polynomial systems [23]. For the set of parameters from challenge A, we thus have to solve systems of 18 quadratic equations in 20 variables, and since they are underdetermined, we can specialize two variables without loss of generality. These systems can be solve either directly or with the hybrid approach [17]; in the first case, our F4 variant will be relevant only if one wants to break several different instances of the MinRank problem.

Experiments with F4 and our variant show that, either for the full systems or the systems with one specialized variable, the matrices involved at different steps are quite large (up to 39138 × 22968) and relatively sparse (less than 5% non-zero entries). With both types of systems, a lot of reductions to zero occurs; for example, we have observed that for the full system at the 8th step, 17442 critical pairs among 17739 reduce to zero. This makes it clear that the classic F4 algorithm is not well suited for these specific systems.

It is difficult to compare our timings with those given in [17] using F5: besides the fact that the experiments were executed on different computers, the linear algebra used in Faugère's FGb implementation of F5 (whose source code is not public) seems to be highly optimized, even more so than in Magma's implementation of F4. On this point, our own implementation is clearly not competitive: for example, at the 7th step for the full system, Magma's F4 reduces a 26723 × 20223 matrix in 28.95 sec, whereas at the same step our implementation reduces a slightly smaller matrix of size 25918 × 19392 in 81.52 sec. Despite these limitations, we have obtained timings comparable with those of [17], listed in the table below. This means that with a more elaborate implementation of linear algebra, our F4 variant would probably be the most efficient for these systems.

	F5	F4Remake	F4	F4 Magma
full system	30.0	27.87	320.2	116.6
1 specialized variable	1.85	2.605	9.654	3.560

Fig. 2. Experimental results on MinRank

Computations were executed on a Xeon bi-processor 3.2 GHz for F5. The results of `F4Remake` have been obtained after a precomputation over $\mathbb{F}_{257}$ of 4682 sec for the full system and 113 sec for the system with one variable specialized.

4.4 Katsura benchmarks

To illustrate the approach presented in section 3.2, we have applied our algorithm to the computation of the Gröbner bases of the Katsura11 and Katsura12 systems [21], over two prime fields of size 16 and 32 bits. As already explained, the idea is to run a precomputation on a small prime field before executing `F4Remake` over a large field (actually, for Katsura12 the first prime $p = 251$ we chose was weakly unlucky). The timings show that for both systems, the speed gain on 32 bits compensates the precomputation overhead, contrarily to the 16 bits case.

	8 bits	16 bits			32 bits		
	Precomputation	F4Remake	F4	F4 Magma	F4Remake	F4	F4 Magma
Katsura11	27.83	9.050	31.83	19.00	15.50	60.93	84.1
Katsura12	202.5	52.66	215.4	143.3	111.4	578.8	$> 5\,h$

Fig. 3. Experimental results on Katsura11 and Katsura12

As a side note, we observed that surprisingly, the matrices created by F4 are quite smaller in our version than in Magma (e.g. 15393 × 19368 versus 20162 × 24137 at step 12 of Katsura12); of course, both version still find the same new polynomials at each step. This phenomenon was already present in the previous systems, but not in such a proportion. This seems to indicate that our implementation of the `Simplify` subroutine is much more efficient.

5 Conclusion

We have presented in this article a variant of the F4 algorithm that provides a very efficient probabilistic method for computing Gröbner bases; it is especially designed for the case where many similar polynomial systems have to be solved. We have given a precise analysis of this context, estimated the probability of success, and evaluated both theoretically and experimentally the performances of our algorithm, showing that it is well adapted for algebraic attacks on cryptosystems.

Since Faugère's F5 algorithm is considered as the most efficient tool for computing Gröbner bases, we have tried as much as possible to compare its performances with our F4 variant. Clearly, F5 remains irreplaceable when the Gröbner basis of only one system has to be computed or when the base field is too small, in particular over $\mathbb{F}_2$. However, our method should be used preferentially as soon as several Gröbner bases have to be computed and the base field is large enough for the considered family of systems. The obtained timings support in part this claim, indicating that with a more elaborate implementation of linear algebra our algorithm would outperform F5 in most cases.

Acknowledgements: The authors would like to thank Jean-Charles Faugère for pointing out existing previous works, and Daniel Lazard for his helpful remarks.

References

1. D. Augot, M. Bardet, and J.-C. Faugère. On the decoding of binary cyclic codes with the Newton identities. *J. Symbolic Comput.*, 44(12):1608–1625, 2009.
2. G. Bard. *Algebraic Cryptanalysis.* Springer-Verlag, New York, first edition, 2009.
3. M. Bardet, J.-C. Faugère, B. Salvy, and B.-Y. Yang. Asymptotic behaviour of the degree of regularity of semi-regular polynomial systems. Presented at MEGA'05, Eighth International Symposium on Effective Methods in Algebraic Geometry, 2005.
4. L. Bettale, J.-C. Faugère, and L. Perret. Hybrid approach for solving multivariate systems over finite fields. *Journal of Mathematical Cryptology*, pages 177–197, 2009.
5. W. Bosma, J. J. Cannon, and C. Playoust. The Magma algebra system I: The user language. *J. Symb. Comput.*, 24(3/4):235–265, 1997.
6. B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal.* PhD thesis, University of Innsbruck, Austria, 1965.
7. B. Buchberger. A criterion for detecting unnecessary reductions in the construction of Gröbner bases. In E. W. Ng, editor, *Proc. of the EUROSAM 79*, volume 72 of *Lecture Notes in Computer Science*, pages 3–21. Copyright: Springer, Berlin - Heidelberg - New York, 1979.
8. B. Buchberger. Gröbner bases: An algorithmic method in polynomial ideal theory. In N. Bose, editor, *Multidimensional systems theory, Progress, directions and open problems, Math. Appl. 16*, pages 184–232. D. Reidel Publ. Co., 1985.
9. N. Courtois. Efficient zero-knowledge authentication based on a linear algebra problem MinRank. In *Advances in Cryptology – ASIACRYPT 2001*, pages 402–421. Springer, 2001.
10. N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *Advances in Cryptology – EUROCRYPT 2000*, pages 392–407. Springer, 2000.
11. C. Diem. On the discrete logarithm problem in elliptic curves. Preprint, available at: `http://www.math.uni-leipzig.de/~diem/preprints/dlp-ell-curves.pdf`, 2009.
12. G. L. Ebert. Some comments on the modular approach to Gröbner-bases. *SIGSAM Bull.*, 17(2):28–32, 1983.

13. C. Eder and J. Perry. F5C: a variant of Faugère's F5 algorithm with reduced Gröbner bases. arXiv/0906.2967, 2009.
14. J.-C. Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, June 1999.
15. J.-C. Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *Proceedings of ISSAC 2002*, New York, 2002. ACM.
16. J.-C. Faugère and A. Joux. Algebraic cryptanalysis of hidden field equation (HFE) cryptosystems using Gröbner bases. In *CRYPTO*, pages 44–60, 2003.
17. J.-C. Faugère, F. Levy-Dit-Vehel, and L. Perret. Cryptanalysis of MinRank. In *Advances in Cryptology – CRYPTO 2008*, pages 280–296, Berlin, Heidelberg, 2008. Springer-Verlag.
18. P. Gaudry. Index calculus for abelian varieties of small dimension and the elliptic curve discrete logarithm problem. *J. Symbolic Computation*, 2008. doi:10.1016/j.jsc.2008.08.005.
19. R. Gebauer and H. M. Möller. On an installation of Buchberger's algorithm. *J. Symbolic Comput.*, 6(2-3):275–286, 1988.
20. A. Joux and V. Vitse. Elliptic curve discrete logarithm problem over small degree extension fields. Application to the static Diffie–Hellman problem on $E(\mathbb{F}_{q^5})$. Cryptology ePrint Archive, Report 2010/157, 2010.
21. S. Katsura, W. Fukuda, S. Inawashiro, N. M. Fujiki, and R. Gebauer. Distribution of effective field in the Ising spin glass of the $\pm J$ model at $T = 0$. *Cell Biochem. Biophys.*, 11(1):309–319, 1987.
22. A. Kipnis, J. Patarin, and L. Goubin. Unbalanced Oil and Vinegar signature schemes. In *Advances in Cryptology – EUROCRYPT'99*, pages 206–222. Springer, 1999.
23. A. Kipnis and A. Shamir. Cryptanalysis of the HFE public key cryptosystem by relinearization. In *Advances in Cryptology – CRYPTO' 99*, pages 19–30. Springer Berlin, Heidelberg, 1999.
24. D. Lazard. Gröbner bases, Gaussian elimination and resolution of systems of algebraic equations. In *Computer algebra (London, 1983)*, volume 162 of *Lecture Notes in Comput. Sci.*, pages 146–156. Springer, Berlin, 1983.
25. F. Macaulay. Some formulae in elimination. *Proceedings of London Mathematical Society*, pages 3–38, 1902.
26. M. S. E. Mohamed, W. S. A. E. Mohamed, J. Ding, and J. Buchmann. MXL2: Solving polynomial equations over $GF(2)$ using an improved mutant strategy. In *PQCrypto*, pages 203–215. Springer, 2008.
27. T. Sasaki and T. Takeshima. A modular method for Gröbner-basis construction over $\mathbb{Q}$ and solving system of algebraic equations. *J. Inf. Process.*, 12(4):371–379, 1989.
28. I. Semaev. Summation polynomials and the discrete logarithm problem on elliptic curves. Cryptology ePrint Archive, Report 2004/031, 2004.
29. C. Traverso. Gröbner trace algorithms. In *Symbolic and algebraic computation (Rome, 1988)*, volume 358 of *Lecture Notes in Comput. Sci.*, pages 125–138. Springer, Berlin, 1989.
30. V. Weispfenning. Comprehensive Gröbner bases. *J. Symbolic Comput.*, 14(1):1–29, 1992.

Improved Agreeing-Gluing Algorithm

Igor Semaev

Department of Informatics, University of Bergen, Norway
igor@ii.uib.no

Abstract. A system of algebraic equations over a finite field is called sparse if each equation depends on a low number of variables. Finding efficiently solutions to the system is an underlying hard problem in the cryptanalysis of modern ciphers. In this paper a deterministic Improved Agreeing-Gluing Algorithm is introduced. The expected running time of the new Algorithm on uniformly random instances of the problem is rigorously estimated. The estimate is at present the best theoretical bound on the complexity of solving average instances of the problem. In particular, this is a significant improvement over those in our earlier papers [15, 17]. In sparse Boolean equations a gap between the worst case and the average time complexity of the problem has significantly increased. Also we formulate *Average Time Complexity Conjecture*. If proved that will have far-reaching consequences in the field of cryptanalysis and in computing in general.

1 Introduction

1.1 The problem and motivation

Let (q, l, n, m) be a quadruple of natural numbers, where q is a prime power. Then F_q denotes a finite field with q elements and $X = \{x_1, x_2, \ldots, x_n\}$ is a set of variables from F_q. By X_i, $1 \le i \le m$ we denote subsets of X of size $l_i \le l$. The system of equations

$$f_1(X_1) = 0, \ldots, f_m(X_m) = 0 \tag{1}$$

is considered, where f_i are polynomials over F_q and they only depend on variables X_i. Such equations are called l-sparse. A solution to (1) over F_q is an assignment in F_q to variables X that satisfies all equations (1). That is a vector of length n over F_q provided the variables X are ordered. The main goal is to find all solutions over F_q. A deterministic Improved Agreeing-Gluing (IAG) Algorithm is suggested. This is presented by two variations. The expected complexity of one variation is rigorously estimated assuming uniform distribution on the problem instances; see Section 1.2. The results provide a significant improvement over earlier average time complexity estimates [15, 17].

The approach, which exploits the sparsity of equations and doesn't depend on their algebraic degree, was studied in [21, 13, 15, 17]. These are guess-and-determine algorithms. In sparse equations the number of guesses on a big enough variable set Y and the time to produce them is much lower than $q^{|Y|}$ due to the Search Algorithm; see Section 8. Previously, no preference was made on which variables to guess. We now argue that guessing values of some particular variables leads to better asymptotic complexity bounds.

The article was motivated by applications in cryptanalysis. Modern ciphers are product, the mappings they implement are compositions of not so many functions in a low number of variables. The similar is true for asymmetric ciphers. Intermediate variables are introduced to simplify equations, describing the cipher, and to get a system of sparse equations. For a more general type of sparse equations, Multiple Right Hand Side linear equations describing in particular AES; see [14]. An efficient solution of the equations breaks the cipher.

Let Y be an ordered string of variables and a be an F_q-vector of the same length. We say that a is a vector in variables Y, or Y-vector, if the entries of a may be assigned to the variables Y, for instance, in case of fixation.

1.2 Probabilistic model

We look for the set of all solutions to (1) over F_q, so we only consider for f_i polynomials of degree at most $q-1$ in each variable. Obviously, the equation $f_i(X_i) = 0$ is determined by the pair (X_i, V_i), where V_i is the set of X_i-vectors, where f_i is zero. Given q, n, m, and $l_1, \ldots, l_m \leq l$, uniform distribution on instances is assumed. As any particular information on equations is beforehand assumed unknown, this looks the most fair probabilistic model to compute expected complexities. The uniformity means

1. the equations in (1) are independently generated. Each equation $f_i(X_i) = 0$ is determined by
2. the subset X_i of size l_i taken uniformly at random from the set of all possible l_i-subsets of X, that is with the probability $\binom{n}{l_i}^{-1}$,
3. and the polynomial f_i taken uniformly at random and independently of X_i from the set of all polynomials of degree $\leq q-1$ in each of variables X_i. In other words, with the equal probability $q^{-q^{l_i}}$.

Running time of any deterministic solving algorithm is a random variable under that model. We assume that m/n tends to $d \geq 1$ as q and l are fixed and n tends to infinity.

Table 1. Algorithms' running time: $q = 2$ and $m = n$.

l	3	4	5	6
the worst case, [10]	1.324^n	1.474^n	1.569^n	1.637^n
Gluing1, expectation, [15]	1.262^n	1.355^n	1.425^n	1.479^n
Gluing2, expectation, [15]	1.238^n	1.326^n	1.393^n	1.446^n
Agreeing-Gluing, expectation, [17]	1.113^n	1.205^n	1.276^n	1.334^n
r	2	3	3	4
Weak Improved Agreeing-Gluing, expectation	1.029^n	1.107^n	1.182^n	1.239^n

.

2 Previous Ideas and the New Approach

One earlier method [15] is based on subsequent computing solutions U_k to the equation subsystems: $f_1(X_1) = 0, \ldots, f_k(X_k) = 0$ for $k = 1, \ldots, m$. Gluing procedure extends instances U_k to instances U_{k+1} by walking throughout a search tree. In the end, all system solutions are U_m. The running time is determined by the maximal of $|U_k|$. Gluing2 is a time-memory trade-off variation of the basis Gluing1 Algorithm. See Table 1 for their running time expectation in case of n Boolean equations in n variables and a variety of l. Any instance of (1) may be encoded by a CNF formula with the clause length of at most $k = \lceil \log_2 q \rceil \, l$ and in $\lceil \log_2 q \rceil \, n$ Boolean variables. Therefore worst case complexity bounds in [10] for k-SAT are also worst case bounds for equations (1). In Boolean case they are shown in the first line of Table 1.

In Agreeing-Gluing Algorithm [17] we only extend those intermediate solutions from U_k that do not contradict with the rest of the equations $f_{k+1}(X_{k+1}) = 0, \dots, f_m(X_m) = 0$. That makes lots of search tree branches cut and implies a better average time complexity.

Let Z_r denote variables that occur in at least r equations (1). The new method has two variations. In the Strong IAG Algorithm the largest r, where Z_r is not empty, is taken. Then Z_r-vectors that do not contradict any of (1) are generated by the Search Algorithm; see Section 8. We denote them W_r. For each $a \in W_r$ the variables Z_r are substituted by the entries of a. New l-sparse equations in a smaller variable set $X \setminus Z_r$ are to solve. One then recursively computes $W_{r-1}, \dots, W_2$. All system solutions are then easy to deduce; see Lemma 2 below.

In the Weak IAG Algorithm r is a parameter. The vectors W_r are generated by the Search Algorithm. The variables Z_r are substituted by the entries of $a \in W_r$. New equations, in case $r \geq 3$, are encoded by a CNF formula and local search algorithm [6] is applied to find all solutions. In this paper we study only the case $l_i = l$. Two last lines in Table 1 show expected complexity of the Weak IAG Algorithm and the optimal value of r. The expected complexity of the Strong IAG Algorithm is not presented in this article. The Agreeing-Gluing Algorithm [17] is a particular case of the present method for $r = 1$.

3 Trivially unsolvable equations

The probability that a randomly chosen equation in l variables is solvable over F_q, i.e., admits at least one solution over F_q, is $1-(1-\frac{1}{q})^{q^l}$. So the probability the equation system (1) is trivially unsolvable(at least one of the equations has no solutions over F_q) is $1-\left[1-(1-\frac{1}{q})^{q^l}\right]^m$. This value tends to 1 as l and q are fixed and $m = dn$ tends to infinity. It is very easy to recognize, with some average complexity R, a trivially unsolvable equation system. However, for small d that only gives a negligible contribution to the average complexity estimate while it is exponential. Let Q denote average complexity of a deterministic algorithm on all instances of (1). Let Q_1 denote average complexity of the algorithm on the instances of (1) which are not trivially unsolvable, i.e., each equation has at least one solution over F_q. In both cases uniform distribution is assumed. By the conditional expectation formula,

$$Q = \left[1-(1-\frac{1}{q})^{q^l}\right]^{dn} Q_1 + \left(1-\left[1-(1-\frac{1}{q})^{q^l}\right]^{dn}\right) R.$$

Therefore, $Q_1 < \left[1-(1-\frac{1}{q})^{q^l}\right]^{-dn} Q$. For $q = 2$ and $d = 1$ that will affect the bound at $l = 3$: in case of the Weak IAG Algorithm, Q_1 becomes bounded by 1.033^n. For all other l the influence is negligible: estimates for Q and Q_1 are almost identical. For larger $d = 1+\delta$ the contribution is larger, but Q becomes sub-exponential fast. So Q_1 remains bounded by a very low exponential function at least for low δ. In fact, we believe that Q_1 becomes sub-exponential too, though it is not proved here.

4 Average time complexity conjecture

The problem of solving (1) is NP-hard as it is polynomially equivalent to k-SAT problem for $k = \lceil \log_2 q \rceil\, l$ and $\lceil \log_2 q \rceil\, n$ variables. A drastic improvement over last few years in average time complexity of solving (1) might indicate

(1) *There exists an algorithm whose expected time complexity on uniformly random instances is sub-exponential in n as q and l are fixed, $m \geq n$ while n tends to infinity.*

We call this statement *Average Time Complexity Conjecture*. If proved that will have far-reaching consequences in the field of cryptanalysis and in computing in general. Previously, for quadratic semi-regular Boolean equation systems it was shown that Gröbner basis algorithm is of sub-exponential complexity provided $n = o(m)$; see [1]. The present conjecture claims that is true for average sparse equation systems regardless their regularity and algebraic degree, and for any $m \geq n$.

5 Related Methods

Gröbner basis algorithm was designed to work with general algebraic equation systems over any ground field; see [3, 12, 8, 9]. It may be used to solving them. The running time is bounded by a value proportional to $\binom{n+D}{D}^{\omega}$ ground field operations, where ω is the exponent in matrix multiplication complexity. The parameter D, called regularity degree, is only computed for semi-regular equations as they are defined in [2]. Theoretical complexity of the Gröbner basis algorithms as F4 or F5 on general polynomial equation systems remains unknown. It is also unknown whether an average equation system behaves semi-regularly, though this seems plausible [2].

The estimate simplifies to $\binom{n}{D}^{\omega}$ for any Boolean equations [2]. In case of l-sparse Boolean equation systems each polynomial in (1) admits at most 2^l monomials. Let l be fixed while $n = m$ tending to infinity. Then each row in Macaulay matrices has bounded number of nonzero terms. Wiedemann algorithm [19] may likely be used to do the linear algebra step. We then put $\omega = 2$. Also let $m = n$. The regularity degree for semi-regular Boolean equations in case $n = m$ was estimated as $D \sim \alpha_d n$, where α_d depends on the equations maximal algebraic degree d. So that $\alpha_2 = 0.09, \alpha_3 = 0.15, \alpha_4 = 0.2$ and so on; see [1]. By estimating the binomial coefficient, the complexity is then $2^{2H(\alpha_d)n}$ up to a polynomial factor, where $H(\alpha)$ is the binary entropy function. One can see that only for quadratic semi-regular polynomials the running time is lower than 2^n, brute force complexity, and equal to 1.832^n bit operations. The best heuristic bound is then of order 1.724^n [20], where the method was combined with variable guessing.

In contrast to [1, 2, 20], when it comes to average equation systems, our estimates are rigorous mathematical statements and very low exponential functions themselves even in non quadratic case; see Table 1. Table 2 presents extended data on IAG algorithm behavior. It shows c_l for a larger variety of $l \leq 19$, where the expected complexity of the IAG Algorithm(computed at $r = 2$) is c_l^n for Boolean l-sparse equations. That compares favorably with the above estimates by Gröbner basis algorithms.

Sparse equations may be encoded by a CNF formula and solved with a SAT-solving software. The asymptotical complexity of modern SAT-solvers, as MiniSat [16], is unknown, though they may be fast in practice [5] for relatively low parameters.

Table 2. IAG Algorithm($r = 2$) base constant c_l, $q = 2$ and $m = n$

l	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
c_l	1.029	1.118	1.191	1.252	1.304	1.347	1.385	1.418	1.448	1.474	1.497	1.518	1.538	1.555	1.571	1.586	1.600

6 Notation, Basic Lemma and Example

Let $Y \subseteq X$ be a subset of variables and a be a Y-vector. Assume $f_i(X_i) = 0$ is one of the equations, where V_i is the set of its local solutions in variables X_i. Let $X_i \not\subseteq Y$ and $b \in V_i$ agree(have the same values) with a on common variables $X_i \cap Y$. Then $V_i(a)$ denotes the projections of such b to variables $X_i \setminus Y$. In other words, $V_i(a)$ are solutions in variables $X_i \setminus Y$ to $f_i(X_i) = 0$ after the Y get substituted by a. If $X_i \subseteq Y$ and the projection of a to X_i does not appear in V_i, then we write $V_i(a) = \emptyset$. Otherwise, $V_i(a) \neq \emptyset$. It is obvious that $V_i(a) = \emptyset$ iff the fixation of Y by constants a contradicts the equation $f_i(X_i) = 0$. The following statement is the basement for the IAG Algorithm complexity analysis in Section 10.

Lemma 1. *Let W be the set of Y-vectors that contradict none of equations* (1). *Assume that $X_1, X_2, \dots, X_m$ are fixed and polynomials $f_1, f_2, \dots, f_m$ are taken uniformly at random as in Section* 1.2. *Then the expectation of $|W|$ is given by*

$$\boldsymbol{E}_{f_1,\dots,f_m}|W| = q^{|Y|} \prod_{i=1}^{m} \left(1 - (1 - \frac{1}{q})^{q^{|X_i \setminus Y|}}\right).$$

Proof. Let a be a Y-vector. We compute $\mathbf{Pr}(a \in W)$, the probability that a contradicts non of $f_i(X_i) = 0$. As f_i are independent,

$$\mathbf{Pr}(a \in W) = \prod_{i=1}^{m} \mathbf{Pr}(V_i(a) \neq \emptyset).$$

One sees $\mathbf{Pr}(V_i(a) = \emptyset) = (1 - \frac{1}{q})^{q^{|X_i \setminus Y|}}$ and this value doesn't depend on a. So

$$\mathbf{E}_{f_1,\dots,f_m}|W| = \sum_a \mathbf{Pr}(a \in W) = q^{|Y|}\mathbf{Pr}(a \in W) = q^{|Y|} \prod_{i=1}^{m} \left(1 - (1 - \frac{1}{q})^{q^{|X_i \setminus Y|}}\right).$$

That proves the Lemma. □

According to Section 1.2, $X_1, X_2, \dots, X_m$ are random subsets in X. Therefore the full expectation of $|W|$ is given by

$$\mathbf{E}|W| = \mathbf{E}_{X_1,\dots,X_m}(\mathbf{E}_{f_1,\dots,f_m}|W|) = \mathbf{E}_{X_1,\dots,X_m}\left(q^{|Y|} \prod_{i=1}^{m} \left(1 - (1 - \frac{1}{q})^{q^{|X_i \setminus Y|}}\right)\right) \qquad (2)$$

by Lemma 3 below. This expectation is estimated in Section 10 for $Y = Z_r(k)$, the set of variables that appear in at least $r \geq 1$ of $X_1, \dots, X_k$, where all $l_i = l$. Therefore, $Y = Z_r(k)$ is here a random variable too. The final estimate is (14). Its maximum in k upper bounds the first stage expected complexity of the Weak IAG Algorithm asymptotically. The second stage complexity is similarly estimated in Section 11.

We have $Z_r(r) \subseteq Z_r(r+1) \subseteq \ldots \subseteq Z_r(m) = Z_r$. Let $W_r(k)$ be the set of $Z_r(k)$-vectors a such that $V_i(a) \neq \emptyset$ for all $i = 1, \ldots, m$. The Search Algorithm, in Sections 7 and 8, extends instances $W_r(k)$ to $W_r(k+1)$. Its output is $W_r = W_r(m)$. Three cases should be studied separately.

Case $r = 1$**.** Then $U_m = W_1$, all system solutions in variables $X_1 \cup \ldots \cup X_m$. Extending $W_1(k)$ to $W_1(k+1)$ by walking over a search tree is the Agreeing-Gluing Algorithm [17].

Case $r = 2$**.** Remark that variables in different $X_i \setminus Z_2$ are pairwise different.

Lemma 2. *Let $X_{i_1}, X_{i_2}, \ldots, X_{i_s}$ be all variable sets such that $X_{i_j} \not\subseteq Z_2$. After reordering of variables it holds that*

$$U_m = \bigcup_{a \in W_2} \{a\} \times V_{i_1}(a) \times V_{i_2}(a) \ldots \times V_{i_s}(a). \tag{3}$$

Example Let the system of three Boolean equations be given:

x_1	x_2	x_3
0	0	1
1	0	0
1	1	1
1	0	1

,

x_3	x_4	x_5
0	0	0
1	0	1
1	1	1
0	0	1

,

x_5	x_6	x_7
0	0	0
0	1	1
1	1	0
1	0	1

.

Then $Z_2(2) = \{x_3\}$ and $Z_2(3) = \{x_3, x_5\}$ and the directed products (3) are:

x_3	x_5	x_1	x_2		x_4		x_6	x_7
0	0	1	0	$\times$	0	$\times$	0	0
							1	1
0	1	1	0	$\times$	0	$\times$	1	0
							0	1
1	1	0	0	$\times$	0	$\times$	1	0
		1	1		1		0	1
		1	0					

.

So 16 solutions to the system are represented by three strings of length 2, that is by $0, 0$, and $0, 1$, and $1, 1$ related to variables x_3, x_5.

Case $r \geq 3$**.** The Search Algorithm returns some $a \in W_r$. The variables Z_r are substituted by the entries of a. The problem is represented in a conjunctive normal form (CNF) with the clause length of at most $k = \lceil \log_2 q \rceil\, l$ and in $n_1 = \lceil \log_2 q \rceil\, |X \setminus Z_r|$ Boolean variables. Local search algorithm, described in [6], is used to find all solutions. In worst case, that takes $O(N(2 - \frac{2}{k+1})^{n_1})$ bit operations to find N solutions.

7 Weak IAG Algorithm

We define a rooted search tree. The tree has at most $m+1$ levels numbered $0, 1, \ldots, m$. The root at level 0 is labeled by $\emptyset$. Vertices at level k are labeled by vectors $W_r(k) \neq \emptyset$ or $\emptyset$ if $Z_r(k) = \emptyset$.

Let a be a level k vertex. It is connected to a level $k+1$ vertex b whenever a is a sub-vector of b. Remark that $Z_r(k) \subseteq Z_r(k+1)$. We now describe the Algorithm.

Stage 1(Search Algorithm) It starts at the root. Let the Algorithm be at a level k vertex a which is extended to b with a projection of $V_{k+1}(a)$ to variables $Z_r(k+1) \setminus Z_r(k)$. If b does not contradict to any of the equations , then b is a level $k+1$ vertex. The Algorithm walks to that. Otherwise, another projection is taken to extend a. If all the projections are exhausted, the Algorithm backtracks to level $k-1$. This stage output is $W_r = W_r(m)$.

Stage 2 Let the Algorithm achieve a vertex $a \in W_r$. If $r = 1$, then a is a system solution. If $r = 2$, then the system solutions are deduced with (3). If $r \geq 3$, a system of l-sparse equations in variables $X \setminus Z_r$ after substituting Z_r by constants a is solved with local search. If no vertex at level m is hit , then the system has no solution.

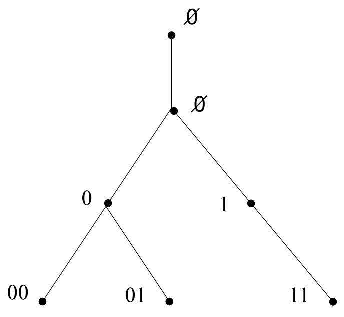


Fig. 1. The search tree.

Theorem 1. *1. If $r < 3$, then the algorithm running time is $O\left(m \sum_i |W_r(i)|\right)$ operations with vectors over F_q of length at most n.*

2. If $r \geq 3$, then the running time is $O\left(m \sum_i |W_r(i)| + |W_r|\, c^{n-|Z_r|}\right)$ operations, where $c = (2 - \frac{2}{l\lceil \log_2 q\rceil + 1})^{\lceil \log_2 q\rceil}$.

The values $|W_r(k)|$ are random under the probabilistic model. They depend on the sets $X_1, \ldots, X_m$ and the polynomials $f_1, \ldots, f_m$. In Section 10 we estimate the maximal of their expectations by using Lemma 1. The expectation of $|W_r|\, c^{n-|Z_r|}$ is estimated in Section 11. For a range of r the estimates are computed with an optimization software like MAPLE. One then finds r such that the running time expectation is minimal. Remark that the computation does not depend on n.

The search tree for the example system is presented in Fig. 1, where $r = 2$. Level 2 vertices are labeled by $W_2(2) = \{(0), (1)\}$, vectors in variables $Z_2(2) = \{x_3\}$. The vertices at level 3 are labeled by $W_2(3) = \{(0,0), (0,1), (1,1)\}$, vectors in variables $Z_2(3) = \{x_3, x_5\}$.

8 General Search Algorithm

Given $Y \subseteq X$, the Algorithm finds all Y-vectors over F_q that do not contradict any of equations (1). A subset sequence $Y_1 \subseteq Y_2 \subseteq \ldots \subseteq Y_s = Y$ is taken. That defines a search tree. The root is labeled by $\emptyset$, the vertices at level $1 \leq k \leq s$ are labeled by Y_k-vectors that do not contradict any of (1). We denote them $W(k)$. Vertices a and b at subsequent levels are connected if a is a sub-vector of b. The algorithm walks with backtracking throughout the tree by constructing instances $W(k)$. The running time is proportional to $|W(1)|q^{|Y_2 \setminus Y_1|} + |W(2)|q^{|Y_3 \setminus Y_2|} + \ldots + |W(s-1)|q^{|Y_s \setminus Y_{s-1}|}$ operations. A sequence of subsets that minimizes the running time may be taken. In IAG Algorithms the sequence is $Z_r(r) \subseteq Z_r(r+1) \subseteq \ldots \subseteq Z_r(m) = Z_r$. In practice, one may find that a Y_k-vector a contradicts the whole system (1) but not only each of the equations taken separately. One then runs the Agreeing Algorithm [14, 18] after the variables Y_k get substituted by constants a. Even if no contradiction is found, one may learn values of some new variables. That improves the method efficiency. However such a variation seems difficult to evaluate.

9 Tools

In this Section we collect miscellaneous auxiliary statements. Let $\eta = \eta(x, y)$ be any variable that depends on two independent discrete random variables x and y. Then $\mathbf{E}_y\,\eta$ denotes the expectation of η, where y is generated to its initial distribution. So $\mathbf{E}_y\,\eta$ is a function in x.

Lemma 3. [17] *For the full expectation of $\eta = \eta(x, y)$ we have*

$$\boldsymbol{E}_{x,y}\,\eta = \boldsymbol{E}_x(\boldsymbol{E}_y(\eta)).$$

Random Allocations Theory studies random allocations of particles(balls) into boxes, see [11]. Let k complexes of particles be independently and uniformly allocated into n boxes, $l_i \leq n$ particles at the i-th allocation. This means that at the i-th allocation any l_i boxes are occupied with the equal probability $\binom{n}{l_i}^{-1}$. This is how variable sets $X_1, \ldots, X_m$ are generated according to Section 1.2. Let $\nu_1, \ldots, \nu_n$ be the string of box frequencies, that is ν_i is the number of particles in the i-th box. Let $A = A(\nu_1, \ldots, \nu_n)$ be any event depending on the frequencies ν_i. Let also $\mathbf{Pr}(A|\, l_1, \ldots, l_k)$ denote the probability of the event A under the allocation by complexes of $l_1, \ldots, l_k$ particles.

Lemma 4.

$$\boldsymbol{Pr}(A|\, l_1, \ldots, l_k) \leq \frac{\boldsymbol{Pr}(A|\, 1, \ldots, 1)}{\prod_{i=1}^{k}(1 - 1/n)\ldots(1 - (l_i - 1)/n)},$$

where $\boldsymbol{Pr}(A|\, 1, \ldots, 1)$ is the probability of A under condition that $L = l_1 + \ldots + l_k$ particles are allocated one after the other, i.e., L of 1's are in the expression $\boldsymbol{Pr}(A|\, 1, \ldots, 1)$.

Proof. Let L particles be independently and uniformly allocated into n boxes one after the other. Let B denote the event that the first l_1 particles were allocated into different boxes, the following l_2 were allocated into different boxes etc, until the last l_k particles were allocated into different boxes. In other words, the event B occurs if the particles are allocated by complexes of size $l_1, l_2, \ldots, l_k$. Then $\mathbf{Pr}(B) = \prod_{i=1}^{k}(1 - 1/n)\ldots(1 - (l_i - 1)/n)$ as the particles were allocated independently. By the complete probability formula we get

$$\begin{aligned}\mathbf{Pr}(A|\, 1, \ldots, 1) &= \mathbf{Pr}(B)\,\mathbf{Pr}(A|\, B) + \mathbf{Pr}(\bar{B})\,\mathbf{Pr}(A|\, \bar{B}) \\ &\geq \mathbf{Pr}(B)\,\mathbf{Pr}(A|\, B) = \mathbf{Pr}(B)\,\mathbf{Pr}(A|\, l_1, \ldots, l_k)\end{aligned}$$

as $\mathbf{Pr}(A|\, B) = \mathbf{Pr}(A|\, l_1, \ldots, l_k)$. That proves the Lemma. □

Let $f^n(z) = \sum_{k=0}^{\infty} a_{n,k}\, z^k$, where real $a_{n,k} \geq 0$, be an analytic function for any natural n.

Lemma 5. *For any real $z_0 > 0$*

$$a_{n,k} \leq \frac{f^n(z_0)}{z_0^k}.$$

Proof. The expansion of f^n has only nonnegative coefficients, so $a_{n,k}\, z_0^k \leq f^n(z_0)$. $\square$

To minimize the estimate one may take a positive root z_0 to

$$\frac{\partial(n \ln f(z) - k\ \ln z)}{\partial z} = 0$$

if there exist any. In case there is only one root, the Lemma estimate is proportional to the main term of the asymptotic expansion for $a_{n,k}$ with saddle point method as n and k tend to infinity; see [4]. Lemma 5 estimate is then asymptotically close to the real value of $a_{n,k}$. We use this observation in Lemmas 6, 7 and 11.

Let $\boldsymbol{\mu}_r = \boldsymbol{\mu}_r(t,n)$ be the number of boxes with just r particle after uniform allocation of t particles one after the other into n boxes. Let $\mathbf{E}\,(x_1^{\boldsymbol{\mu}_{r_1}} \dots x_s^{\boldsymbol{\mu}_{r_s}})$ be the expectation of the random variable $x_1^{\boldsymbol{\mu}_{r_1}} \dots x_s^{\boldsymbol{\mu}_{r_s}}$, where $x_1, \dots, x_s$ are any variables. By definition,

$$\mathbf{E}\,(x_1^{\boldsymbol{\mu}_{r_1}} \dots x_s^{\boldsymbol{\mu}_{r_s}}) = \sum_{k_1,\dots,k_s} \mathbf{Pr}(\boldsymbol{\mu}_{r_1} = k_1, \dots, \boldsymbol{\mu}_{r_s} = k_s)\, x_1^{k_1} \dots x_s^{k_s}.$$

Theorem 2 in Chapter 2, Section 1 of [11] states

$$\sum_{t=0}^{\infty} \frac{n^t\, z^t}{t!} \mathbf{E}\,(x_1^{\boldsymbol{\mu}_{r_1}} \dots x_s^{\boldsymbol{\mu}_{r_s}}) = \left(e^z + \frac{z^{r_1}}{r_1!}(x_1 - 1) + \dots + \frac{z^{r_s}}{r_s!}\,(x_s - 1)\right)^n. \tag{4}$$

In particular, we get

$$\sum_{t=0}^{\infty} \frac{n^t\, z^t}{t!} \mathbf{E}\,(x_0^{\boldsymbol{\mu}_0} \dots x_{r-1}^{\boldsymbol{\mu}_{r-1}}) = \left(e^z + (x_0 - 1) + \dots + \frac{z^{r-1}}{(r-1)!}\,(x_{r-1} - 1)\right)^n.$$

We there put $x_0 = \dots = x_{r-1} = 0$ and get

$$\left(e^z - 1 - z \dots - \frac{z^{r-1}}{(r-1)!}\right)^n = \sum_{t=nr}^{\infty} \frac{n^t z^t}{t!}\, \mathbf{Pr}(\boldsymbol{\mu}_0 = 0, \dots, \boldsymbol{\mu}_{r-1} = 0)$$

as $\mathbf{Pr}(\boldsymbol{\mu}_0 = 0, \dots, \boldsymbol{\mu}_{r-1} = 0) = 0$ for $t < nr$. Let $g(z) = e^z - 1 - z \dots - \frac{z^{r-1}}{(r-1)!}$.

Lemma 6. *Let $r \geq 1$. For any natural number $t \geq nr$*

$$\boldsymbol{Pr}(\boldsymbol{\mu}_0(t,n) = 0, \dots, \boldsymbol{\mu}_{r-1}(t,n) = 0) \leq \frac{g^n(z_0)\; t!}{z_0^t\; n^t},$$

where z_0 is the only nonnegative root of the equation $n\frac{z\left(e^z - 1 - z \dots - \frac{z^{r-2}}{(r-2)!}\right)}{e^z - 1 - z \dots - \frac{z^{r-1}}{(r-1)!}} = t.$

Proof. Let $t > nr$. The equation has the only positive solution z_0. The statement is true by Lemma 5. Let $t = nr$, then $z_0 = 0$. One sees that $\frac{g^n(z_0)\,t!}{x_0^t\,n^t}$ is defined at $z_0 = 0$ and equal to $\frac{(n\,r)!}{(r!)^n\,n^{n\,r}}$. On the other hand, one directly computes

$$\mathbf{Pr}(\boldsymbol{\mu}_0(nr, n) = 0, \ldots, \boldsymbol{\mu}_{r-1}(nr, n) = 0) = \frac{(nr)!}{(r!)^n\,n^{nr}}.$$

The statement is true for any $t \geq nr$. That proves the Lemma. □

It follows from (4) that

$$\sum_{t=0}^{\infty} \frac{n^t\,z^t}{t!} \mathbf{E}\,(x_1^{\boldsymbol{\mu}_1} \ldots x_{r-1}^{\boldsymbol{\mu}_{r-1}}) = \left(e^z + z(x_1 - 1) + \ldots + \frac{z^{r-1}}{(r-1)!}\,(x_{r-1} - 1)\right)^n. \tag{5}$$

Substitute $x_i = x^i$ for $i = 1, \ldots, r-1$. Then

$$\sum_{t=0}^{\infty} \frac{n^t\,z^t}{t!} \mathbf{E}\,(x^{\boldsymbol{\mu}_1 + 2\boldsymbol{\mu}_2 + \ldots + (r-1)\boldsymbol{\mu}_{r-1}})$$
$$= \left[e^z - \left(z + \ldots + \frac{z^{r-1}}{(r-1)!}\right) + \left(zx + \ldots + \frac{(zx)^{r-1}}{(r-1)!}\right)\right]^n.$$

We have

$$\mathbf{E}\,(x^{\boldsymbol{\mu}_1 + 2\boldsymbol{\mu}_2 + \ldots + (r-1)\boldsymbol{\mu}_{r-1}}) = \sum_{k=0}^{t} x^k\,\mathbf{Pr}\,(\boldsymbol{\mu}_1 + 2\boldsymbol{\mu}_2 + \ldots + (r-1)\boldsymbol{\mu}_{r-1} = k)$$

because $\mathbf{Pr}\,(\boldsymbol{\mu}_1 + 2\boldsymbol{\mu}_2 + \ldots + (r-1)\boldsymbol{\mu}_{r-1} = k) = 0$ if $k > t$. We again denote zx by x and get from the last two identities that

$$\sum_{t \geq k} \frac{n^t\,z^{t-k}\,x^k}{t!} \mathbf{Pr}\,(\boldsymbol{\mu}_1 + 2\boldsymbol{\mu}_2 + \ldots + (r-1)\boldsymbol{\mu}_{r-1} = k)$$
$$= \left[e^z - \left(z + \ldots + \frac{z^{r-1}}{(r-1)!}\right) + \left(x + \ldots + \frac{x^{r-1}}{(r-1)!}\right)\right]^n.$$

We now put $z = 0$. Therefore,

$$\left(1 + x + \ldots + \frac{x^{r-1}}{(r-1)!}\right)^n = \sum_{t=0}^{(r-1)n} \frac{n^t\,x^t}{t!} \mathbf{Pr}\,(\boldsymbol{\mu}_1 + 2\boldsymbol{\mu}_2 + \ldots + (r-1)\boldsymbol{\mu}_{r-1} = t).$$

We remark that the probability is zero if $t > (r-1)n$. Let $h(x) = 1 + x \ldots + \frac{x^{r-1}}{(r-1)!}$.

Lemma 7. *Let $r \geq 1$. For any natural number t such that $(r-1)n \geq t \geq 0$ we have*

$$\boldsymbol{Pr}(\boldsymbol{\mu}_1 + 2\boldsymbol{\mu}_2 + \ldots + (r-1)\boldsymbol{\mu}_{r-1} = t) \leq \frac{h^n(x_0)}{x_0^t}\,\frac{t!}{n^t},$$

where x_0 is the only nonnegative root (including ∞) of the equation $n\,\frac{x\left(1 + x + \ldots + \frac{x^{r-2}}{(r-2)!}\right)}{1 + x + \ldots + \frac{x^{r-1}}{(r-1)!}} = t.$

Proof. Let $(r-1)n > t > 0$. The equation has the only positive solution x_0. The estimate is true by Lemma 5. Let $t = 0$, then $x_0 = 0$ and the Lemma is true as both the sides of the inequality are 1. Let $t = (r-1)n$, then $x_0 = \infty$. The right hand side of the inequality is defined at $x_0 = \infty$ and equal to $\frac{t!}{((r-1)!)^n\, n^t}$. On the other hand,

$$\mathbf{Pr}\left(\boldsymbol{\mu}_1 + 2\boldsymbol{\mu}_2 + \ldots + (r-1)\boldsymbol{\mu}_{r-1} = t\right) = \mathbf{Pr}\left(\boldsymbol{\mu}_{r-1}(t,n) = n\right) = \frac{t!}{((r-1)!)^n\, n^t}.$$

That proves the Lemma. □

Lemma 8. *For every integer number $k \geq 0$ it holds that*

$$k^k e^{-k} \leq k! \leq k^k e^{-k}\sqrt{2\pi(k+1)}.$$

10 Complexity Estimate. Stage 1

Let $l_i = l$ for all $i = 1, \ldots, m$. We now estimate the expectation of $|W_r(k)|$. Its maximum in k will be estimated with (14). The following is an implication of Lemma 1 and formula (2).

$$\mathbf{E}|W_r(k)| = \mathbf{E}_{X_1,\ldots,X_m}\left(q^{|Z_r(k)|}\prod_{i=1}^{m}\left(1-(1-\frac{1}{q})^{q^{|X_i\setminus Z_r(k)|}}\right)\right). \tag{6}$$

According to the probabilistic model, $X_1, \ldots, X_m$ are uniformly allocated into the whole variable set X of size n. So we use the language of particle allocation into n boxes from now. In particular, $Z_r(k)$ is the set of boxes with at least r particles after uniform allocation by k complexes of size l. We split the product in (6):

$$\mathbf{E}|W_r(k)| = \mathbf{E}_{X_1,\ldots,X_m}\left(q^{|Z_r(k)|}\prod_{i=1}^{k}\left(1-(1-\frac{1}{q})^{q^{|X_i\setminus Z_r(k)|}}\right)\prod_{j=k+1}^{m}\left(1-(1-\frac{1}{q})^{q^{|X_j\setminus Z_r(k)|}}\right)\right).$$

We say the event $A = A(U, t_1, \ldots, t_k)$ occurs if $Z_r(k) = U$ and $|X_i \setminus U| = t_i$, where $i = 1, \ldots, k$. With the conditional expectation formula we get

$$\mathbf{E}|W_r(k)| = \sum_{U}\sum_{t_1,\ldots,t_k} q^{|U|}\prod_{i=1}^{k}\left(1-(1-\frac{1}{q})^{q^{t_i}}\right)\mathbf{E}(A)\,\mathbf{Pr}(A), \tag{7}$$

where U runs over all subsets of X and $0 \leq t_i \leq l$ and we denoted

$$\mathbf{E}(A) = \mathbf{E}_{X_1,\ldots,X_m}\left(\prod_{j=k+1}^{m}\left(1-(1-\frac{1}{q})^{q^{|X_j\setminus Z_r(k)|}}\right)\middle|\, A\right).$$

We now estimate the probability of the event A. Let $|U| = u$.

Lemma 9. *Let $L = lk$ and $T = t_1 + \ldots + t_k$. Then*

$$\boldsymbol{Pr}(A) \leq \frac{\left(\frac{u}{n}\right)^{L-T}\left(\frac{n-u}{n}\right)^{T} P_1(L-T, u)\; P_2(T, n-u)\; \prod_{i=1}^{k}\binom{l}{t_i}}{\prod_{i=1}^{l-1}(1-\frac{i}{n})^k},$$

where

$$P_1(L-T,u) = \boldsymbol{Pr}(\boldsymbol{\mu}_0(L-T,u)=0,\dots,\boldsymbol{\mu}_{r-1}(L-T,u)=0),$$
$$P_2(T,n-u) = \boldsymbol{Pr}(\boldsymbol{\mu}_1(T,n-u)+2\,\boldsymbol{\mu}_2(T,n-u)+\dots+(r-1)\,\boldsymbol{\mu}_{r-1}(T,n-u)=T).$$

Proof. Assume $0<u<n$, otherwise the statement is easy with Lemma 4. We say the event B occurs if $|X_i \setminus U| = t_i$ for $i=1,\dots,k$. Then $\mathbf{Pr}(A) = \mathbf{Pr}(B)\mathbf{Pr}(A|B)$.

$$\mathbf{Pr}(B) = \prod_{i=1}^{k} \mathbf{Pr}(|X_i \setminus U| = t_i) = \prod_{i=1}^{k} \frac{\binom{u}{l-t_i}\binom{n-u}{t_i}}{\binom{n}{l}} =$$
$$= \prod_{i=1}^{k} \binom{l}{t_i} \left(\frac{u}{n}\right)^{l-t_i} \left(\frac{n-u}{n}\right)^{t_i} \frac{(1-\frac{1}{u})\dots(1-\frac{l-t_i-1}{u})(1-\frac{1}{n-u})\dots(1-\frac{t_i-1}{n-u})}{(1-\frac{1}{n})\dots(1-\frac{l-1}{n})}$$
$$= \frac{\left(\frac{u}{n}\right)^{L-T}\left(\frac{n-u}{n}\right)^{T} \prod_{i=1}^{k}\binom{l}{t_i} \prod_{i=1}^{k}(1-\frac{1}{u})\dots(1-\frac{l-t_i-1}{u}) \prod_{i=1}^{k}(1-\frac{1}{n-u})\dots(1-\frac{t_i-1}{n-u})}{\prod_{i=1}^{l-1}(1-\frac{i}{n})^k}.$$

The event $A|B$ occurs if and only if the following two events A_1 and A_2 occur simultaneously. First, the complexes of $l-t_1,\dots,l-t_k$ particles are allocated into $|U|=u$ boxes, where each box is occupied by at least r particles. Second, the complexes of $t_1,\dots,t_k$ particles are allocated into $|X\setminus U| = n-u$ boxes, where each box is occupied by at most $r-1$ particles. These are independent events. Therefore $\mathbf{Pr}(A|B) = \mathbf{Pr}(A_1)\mathbf{Pr}(A_2)$.

Let $\boldsymbol{\mu}'_s(t_1,\dots,t_k,n)$ be the number of boxes with exactly s particles after k uniform allocations into n boxes by complexes of $t_1,\dots,t_k$ particles. The event A_1 occurs if and only if $\boldsymbol{\mu}'_i(l-t_1,\dots,l-t_k,u)=0$ for $i=0,\dots,r-1$. The event A_2 occurs if and only if $\boldsymbol{\mu}'_i(t_1,\dots,t_k,n-u)=0$ for $i\ge r$. The latter is equivalent to

$$\boldsymbol{\mu}'_1(t_1,\dots,t_k,n-u)+2\,\boldsymbol{\mu}'_2(t_1,\dots,t_k,n-u)+\dots+(r-1)\,\boldsymbol{\mu}'_{r-1}(t_1,\dots,t_k,n-u)=T.$$

By Lemma 4,

$$\mathbf{Pr}(A_1) \le \frac{P_1(L-T,u)}{\prod_{i=1}^{k}(1-\frac{1}{u})\dots(1-\frac{l-t_i-1}{u})}$$

and

$$\mathbf{Pr}(A_2) \le \frac{P_2(T,n-u)}{\prod_{i=1}^{k}(1-\frac{1}{n-u})\dots(1-\frac{t_i-1}{n-u})}$$

So $\mathbf{Pr}(A) = \mathbf{Pr}(B)\mathbf{Pr}(A|B) =$

$$= \mathbf{Pr}(B)\mathbf{Pr}(A_1)\mathbf{Pr}(A_2) \le \frac{\left(\frac{u}{n}\right)^{L-T}\left(\frac{n-u}{n}\right)^{T} P_1(L-T,u)\;P_2(T,n-u)\;\prod_{i=1}^{k}\binom{l}{t_i}}{\prod_{i=1}^{l-1}(1-\frac{i}{n})^k}.$$

That proves the Lemma. □

Lemma 10. $\boldsymbol{E}(A) = \boldsymbol{E}(u)$, *where*

$$\boldsymbol{E}(u) = \prod_{j=k+1}^{m} \boldsymbol{E}_{X_j}\left(1-(1-\frac{1}{q})^{q^{|X_j\setminus U|}}\right)$$

only depends on the size u of the set U.

Proof. For any $X_1, \dots, X_k$, where A occurs, we have

$$\mathbf{E}_{X_{k+1},\dots,X_m}\left(\prod_{j=k+1}^{m}\left(1-(1-\frac{1}{q})^{q^{|X_j\setminus Z_r(k)|}}\right)\middle| A\right) = \prod_{j=k+1}^{m}\mathbf{E}_{X_j}\left(1-(1-\frac{1}{q})^{q^{|X_j\setminus U|}}\right).$$

as $X_{k+1}, \dots, X_m$ are independent. That doesn't depend on $X_1, \dots, X_k$. By Lemma 3, $\mathbf{E}(A) = \mathbf{E}_{X_1,\dots,X_k}(\mathbf{E}_{X_{k+1},\dots,X_m}(\dots| A)) = \mathbf{E}(u)$. The value depends on the size of the set U and not on the set itself. That proves the Lemma. □

From (7) by Lemma 10, we get

$$\mathbf{E}|W_r(k)| = \sum_{u=0}^{n}\binom{n}{u} q^u\ \mathbf{E}(u) \sum_{t_1,\dots,t_k}\prod_{i=1}^{k}\left(1-(1-\frac{1}{q})^{q^{t_i}}\right)\ \mathbf{Pr}(A) \tag{8}$$

as $\mathbf{Pr}(A)$ only depends on $u, t_1, \dots, t_k$. From (8) by Lemma 9,

$$\mathbf{E}|W_r(k)| \le \frac{1}{\prod_{i=1}^{l-1}(1-\frac{i}{n})^k}\sum_{u=0}^{n}\binom{n}{u} q^u\ \mathbf{E}(u) \tag{9}$$
$$\times \sum_{T=0}^{L} C_T\ \left(\frac{u}{n}\right)^{L-T}\left(\frac{n-u}{n}\right)^{T}\ P_1(L-T,u)\ P_2(T,n-u),$$

where $C_T = \sum_{t_1+\dots+t_k=T}\prod_{i=1}^{k}\binom{l}{t_i}\left(1-(1-\frac{1}{q})^{q^{t_i}}\right)$. Let $f(z) = \sum_{t=0}^{l}\binom{l}{t}\left(1-(1-\frac{1}{q})^{q^t}\right) z^t$. It is obvious that $f^k(z) = \sum_{T=0}^{lk} C_T\ z^T$ and

Lemma 11. *For every $0 \le T \le l\,k$ we have $C_T \le \frac{f^k(z_0)}{z_0^T}$, where z_0 is the only nonnegative root(including ∞ for $T = lk$) to the equation* $k\,\frac{\sum_{t=1}^{l} t\binom{l}{t}\left(1-(1-\frac{1}{q})^{q^t}\right) z^t}{\sum_{t=0}^{l}\binom{l}{t}\left(1-(1-\frac{1}{q})^{q^t}\right) z^t} = T.$

We now estimate $\mathbf{E}(u)$. Let $u = \beta n$, then

$$\mathbf{E}_{X_i}(1-(1-\frac{1}{q})^{q^{|X_i\setminus U|}}) = 1-\sum_{t=0}^{l}\frac{\binom{u}{l-t}\binom{n-u}{t}}{\binom{n}{l}}(1-\frac{1}{q})^{q^t} = 1-\sum_{t=0}^{l}\frac{\binom{\beta n}{l-t}\binom{n-\beta n}{t}}{\binom{n}{l}}(1-\frac{1}{q})^{q^t}.$$

By taking $\lim_{n\to\infty}$, we get

Lemma 12. *Let $|U| = \beta n$, where $0 \le \beta \le 1$ as n tends to ∞, then*

$$\boldsymbol{E}_{X_i}(1-(1-\frac{1}{q})^{q^{|X_i\setminus U|}}) = F(\beta) + O(\frac{1}{n}),$$

where $F(\beta) = 1-\sum_{t=0}^{l}\binom{l}{t}\beta^{l-t}(1-\beta)^t(1-\frac{1}{q})^{q^t}$ and $O(\frac{1}{n})$ is uniformly bounded in β.

Lemmas 10 and 12 imply $\mathbf{E}(u) \le (F(\beta)+\epsilon)^{m-k}$, where ϵ is any positive number and n is big enough. Let $L = \alpha n$ and $T = \gamma n$. So $\frac{m-k}{n} = \frac{m}{n} - \frac{\alpha}{l}$. As $\frac{m}{n}$ tends to d, then

$$\mathbf{E}(u) \le (F(\beta)+\epsilon)^{(d-\frac{\alpha}{l})n} \tag{10}$$

for any positive ϵ as n tends to ∞. By Lemma 6, $P_1(L-T,u) \leq \frac{g^u(x_0)\ (L-T)!}{x_0^{L-T}\ u^{L-T}}$, where x_0 is the only nonnegative root of the equation $\beta\ \frac{x\left(e^x-1-x\ldots-\frac{x^{r-2}}{(r-2)!}\right)}{e^x-1-x\ldots-\frac{x^{r-1}}{(r-1)!}} = \alpha-\gamma$. Therefore, by estimating $(L-T)!$ with Lemma 8, we get

$$P_1(L-T,u) \leq \left[\frac{g^{\beta}(x_0)}{x_0^{\alpha-\gamma}}\left(\frac{\alpha-\gamma}{\beta e}\right)^{\alpha-\gamma}+\epsilon\right]^n, \tag{11}$$

for any positive ϵ and big enough n. By Lemma 7, $P_2(T,n-u) \leq \frac{h^{n-u}(y_0)\ T!}{y_0^T\ (n-u)^T}$. Therefore,

$$P_2(T,n-u) \leq \left[\frac{h^{1-\beta}(y_0)}{y_0^{\gamma}}\left(\frac{\gamma}{(1-\beta)e}\right)^{\gamma}+\epsilon\right]^n, \tag{12}$$

for any positive ϵ and all big n, where y_0 is a nonnegative root to $(1-\beta)\ \frac{y\left(1+y+\ldots+\frac{y^{r-2}}{(r-2)!}\right)}{1+y+\ldots+\frac{y^{r-1}}{(r-1)!}} = \gamma$. By Lemma 11,

$$C_T \leq \left(\frac{f^{\frac{\alpha}{l}}(z_0)}{z_0^{\gamma}}\right)^n, \tag{13}$$

where z_0 is the only nonnegative root to $\frac{\alpha}{l}\ \frac{\sum_{t=1}^{l} t\binom{l}{t}\left(1-(1-\frac{1}{q})^{q^t}\right)z^t}{\sum_{t=0}^{l}\binom{l}{t}\left(1-(1-\frac{1}{q})^{q^t}\right)z^t} = \gamma$. We remark that for any positive ϵ bounds (11), (12), (13) and (10) are true simultaneously for all big enough n. Therefore,

$$\mathbf{E}|W_r(k)| \leq \frac{(n+1)(lm+1)}{\prod_{i=1}^{l-1}(1-\frac{i}{n})^m}\ \max\left[\frac{q^{\beta}\ f^{\frac{\alpha}{l}}(z_0)\ g^{\beta}(x_0)\ h^{1-\beta}(v_0)\ (\alpha-\gamma)^{\alpha-\gamma}\ \gamma^{\gamma}}{\beta^{\beta}\ (1-\beta)^{1-\beta}\ (z_0v_0)^{\gamma}\ x_0^{\alpha-\gamma}\ e^{\alpha}}\ F(\beta)^{d-\frac{\alpha}{l}}+\epsilon\right]^n,$$

for any positive ϵ and big enough n, where Lemma 8 was used to bound the binomial coefficient $\binom{n}{u}$. Therefore,

$$\mathbf{E}|W_r(k)| \leq \left[\max\left(\frac{q^{\beta}\ f^{\frac{\alpha}{l}}(z_0)\ g^{\beta}(x_0)\ h^{1-\beta}(y_0)\ (\alpha-\gamma)^{\alpha-\gamma}\ \gamma^{\gamma}}{\beta^{\beta}\ (1-\beta)^{1-\beta}\ (z_0y_0)^{\gamma}\ x_0^{\alpha-\gamma}\ e^{\alpha}}\ F(\beta)^{d-\frac{\alpha}{l}}\right)+\epsilon\right]^n \tag{14}$$

for any positive ϵ and big enough n. The maximum is over $0 \leq \beta \leq 1$ and $0 \leq \gamma \leq \alpha$. We remark that the parameters α, β, γ should satisfy $r\beta \leq \alpha-\gamma$ and $(r-1)(1-\beta) \leq \gamma$, otherwise $P_1(L-T,u) = 0$ or $P_2(T,n-u) = 0$. The complexity of the first stage is upper bounded by the maximum of (14) over $0 \leq \alpha \leq dl$. For any q, l, d, r one computes a maximum of (14) main term, which is a real valued function in three real variables α, β, γ under some restrictions. That may be done with an advanced optimization package like MAPLE.

11 Complexity Estimate. Stage 2

Let $r \geq 3$. Let $W_r = W_r(m)$ and $Z_r = Z_r(m)$. Let $X_1, \ldots, X_m$ be fixed and $f_1, \ldots, f_m$ randomly generated. Then one proves that $\mathbf{E}_{f_1,\ldots,f_m}\left(|W_r|c^{n-|Z_r|}\right)$ is the expected complexity to compute all solutions, where c is defined in Theorem 1. Similarly to (6),

$$\mathbf{E}\left(|W_r|c^{n-|Z_r|}\right) = \mathbf{E}_{X_1,\ldots,X_m}\left(q^{|Z_r|}c^{n-|Z_r|}\prod_{i=1}^{m}\left(1-(1-\frac{1}{q})^{q^{|X_i\setminus Z_r|}}\right)\right).$$

Let $L = lm$. Similarly to (9),

$$\mathbf{E}\left(|W_r|\, c^{n-|Z_r|}\right) \le \frac{1}{\prod_{i=1}^{l-1}(1-\frac{i}{n})^m} \sum_{u=0}^{n} \binom{n}{u} q^u c^{n-u}$$

$$\times \sum_{T=0}^{L} C_T \left(\frac{u}{n}\right)^{L-T} \left(\frac{n-u}{n}\right)^{T} P_1(L-T,u)\, P_2(T,n-u),$$

where $C_T = \sum_{t_1+\ldots+t_m=T} \prod_{i=1}^{m} \binom{l}{t_i} \left(1-(1-\frac{1}{q})^{q^{t_i}}\right)$. Therefore,

$$\mathbf{E}\left(|W_r|\, c^{n-|Z_r|}\right) \le \left[\max\left(\frac{q^\beta c^{1-\beta}\, f^d(z_0)\, g^\beta(x_0)\, h^{1-\beta}(y_0)\, (dl-\gamma)^{dl-\gamma}\, \gamma^\gamma}{\beta^\beta\, (1-\beta)^{1-\beta}\, (z_0y_0)^\gamma\, x_0^{dl-\gamma}\, e^{dl}}\right) + \epsilon\right]^n \qquad (15)$$

for any positive ϵ and big enough n. The maximum is over $0 \le \beta \le 1$ and $0 \le \gamma \le dl$. We remark that the parameters α, β, γ should satisfy $r\beta \le dl - \gamma$ and $(r-1)(1-\beta) \le \gamma$, otherwise $P_1(L-T,u) = 0$ or $P_2(T,n-u) = 0$.

References

1. M. Bardet, J.-C.Faugére, and B. Salvy, *Complexity of Gröbner basis computation for semi-regular overdetermined sequences over F_2 with solutions in F_2,* Research report RR–5049, INRIA, 2003.
2. M. Bardet, J-C. Faugére, B. Salvy and B-Y. Yang, *Asymptotic Behaviour of the Degree of Regularity of Semi-Regular Polynomial Systems*, in MEGA 2005, 15 pages.
3. B. Buchberger, *Theoretical Basis for the Reduction of Polynomials to Canonical Forms,* SIGSAM Bull. 39(1976), 19-24.
4. E.T. Copson, *Asymptotic expansions,* Cambridge University Press, 1965.
5. N. T. Courtois and G. V. Bard, *Algebraic Cryptanalysis of the Data Encryption Standard*, Crypt. ePrint Arch., report 2006/402.
6. E. Dantsin, A. Goerdt, E. A. Hirsch, R. Kannan, J. M. Kleinberg, C. H. Papadimitriou, P. Raghavan, U. Schning,*A deterministic $(2-2/(k+1))^n$ algorithm for k-SAT based on local search.* Theor. Comput. Sci. 289(2002), pp.69–83.
7. N. Eén, N. Sörensson, MiniSat home page, http://minisat.se/
8. J.-C. Faugère, *A new efficient algorithm for computing Grbner bases (F4),* Journal of Pure and Applied Algebra, vol. 139 (1999), pp. 61-88.
9. J.-C. Faugère, *A new efficient algorithm for computing Gröbner bases without reduction to zero (F5),* in ISSAC 2002, pp. 75 – 83, ACM Press, 2002.
10. K. Iwama, *Worst-Case Upper Bounds for kSAT,* The Bulletin of the EATCS, vol. 82(2004), pp. 61–71.
11. V. Kolchin, A. Sevast'yanov, and V. Chistyakov, *Random allocations,* John Wiley & Sons, 1978.
12. D. Lazard, *Gröbner-bases, Gaussian elimination and resolution of systems of algebraic equations*, in EUROCAL 1983, pp. 146–156.
13. H. Raddum, *Solving non-linear sparse equation systems over $GF(2)$ using graphs,* University of Bergen, preprint, 2004.
14. H. Raddum, I. Semaev, *Solving Multiple Right Hand Sides linear equations*, Des. Codes Cryptogr., vol.49 (2008), pp.147–160.
15. I. Semaev, *On solving sparse algebraic equations over finite fields*, Des. Codes Cryptogr., vol. 49 (2008), pp.47–60.
16. N. Eén, N. Sörensson, MiniSat home page, http://minisat.se/
17. I. Semaev, *Sparse algebraic equations over finite fields,* SIAM J. on Comp., vol. 39(2009), pp. 388-409.

18. I. Semaev, *Sparse Boolean equations and circuit lattices*, Des. Codes Cryptogr.(2010), to appear.
19. D. H. Wiedemann, *Solving sparse linear equations over finite fields,* IEEE Trans. Information Theory, vol. 32(1986), pp. 54–62.
20. B.-Y. Yang, J-M. Chen, and N.Courtois, *On asymptotic security estimates in XL and Gröbner bases-related algebraic cryptanalysis,* LNCS 3269, pp. 401–413, Springer-Verlag, 2004.
21. A. Zakrevskij, I. Vasilkova, *Reducing large systems of Boolean equations,* 4th Int.Workshop on Boolean Problems, Freiberg University, September, 21–22, 2000.

PWXL: A Parallel Wiedemann-XL Algorithm for Solving Polynomial Equations over GF(2)

Wael Said Abdelmageed Mohamed[1], Jintai Ding[2], Thorsten Kleinjung[3], Stanislav Bulygin[4], and Johannes Buchmann[1]

[1] TU Darmstadt, FB Informatik
Hochschulstrasse 10, 64289 Darmstadt, Germany
{wael,buchmann}@cdc.informatik.tu-darmstadt.de
[2] Department of Mathematical Sciences, University of Cincinnati, OH 45220, USA
jintai.ding@uc.edu
[3] EPFL IC LACAL, Station 14, CH-1015 Lausanne, Switzerland
thorsten.kleinjung@epfl.ch
[4] Center for Advanced Security Research Darmstadt (CASED)
Stanislav.Bulygin@cased.de

Abstract. The XL algorithm is an algorithm for solving systems of multivariate polynomial equations over finite fields. XL expands the initial system by multiplying it with monomials below a certain degree. XL then linearizes the expanded system and solves the linear system by Gaussian elimination. Since the linear systems that have to be solved are sparse, the Wiedemann algorithm can be applied to reduce the high time and space complexity of Gaussian elimination. The main contribution of this paper is to show how PWXL, a combination of XL and a parallel Wiedemann solver, further reduces the time and space cost. When using n processors, the running time is divided by n and the linear system is distributed over memory of the n processors without the need to store in one place at any time of the computation. By using PWXL, we are able to solve an HFE system of univariate degree 4352 with 34 equations in 34 variables in almost 13 days using 16 processors, which was never done before by any known algebraic solver.

1 Introduction

In this paper we are interested in solving systems of multivariate quadratic polynomial equations over the field GF(2). This problem is called "MQ-problem" and NP-complete [1]. In cryptography, constructing a system of multivariate polynomial equations that defines the secret of a cryptographic primitive and then solving this system to recover that secret information is called algebraic cryptanalysis (attack). These attacks are applicable to a variety of ciphers, block ciphers like AES and Serpent [2], stream ciphers like Toycrypt [3] and E0 [4], and asymmetric cryptosystems like HFE [5]. Moreover, recently it has been used to attack linear RFID Authentication Protocols [6].

Hidden Field Equations (HFE) is a class of asymmetric algorithms that are used for encryption or signatures. The security of this scheme is not proved while this security is related to the problem of solving a system of multivariate polynomial equations over finite fields. In the extended version of [7], Patarin proposed two explicit quadratic HFE challenge signature schemes. The first challenge, HFE(d=96, n=80), is a scheme that gives signatures of length 80 bits using a secrete polynomial of degree 96 over GF(2). The second challenge, HFE(d=4352, n=144), gives signatures of length 144 bits. The hidden polynomial has a degree 4352 over GF(16) with 36 variables and 4 of the 36 equations are not given public. In [5], Faugère and Joux proposed a theoretical attack with a good complexity as well as a very practical method for breaking the first HFE challenge. While Courtois claims the complexity of 2^{63} on HFE challenge 2 [8].

The XL algorithm, which stands for eXtended Linearization, was proposed in [9] as a simple and powerful algorithm for solving overdetermined systems of polynomial equations. The general strategy of XL can be viewed as a combination of bounded degree Gröbner basis and linearization [10]. The main idea of this algorithm is to produce from each original polynomial equation a large number of higher degree polynomial equations by multiplying the original polynomial with all possible monomials up to some bounded degree, then XL linearizes the extended system and solves it using Gaussian elimination.

Indeed, the Macaulay matrix of the linearized extended system has rows representing multiples of original polynomials and columns representing monomials up to a given degree. The number of non-zero elements in each row is bounded by the maximum number of monomials that appear in one original polynomial. Therefore, the whole matrix tends to be sparse. In this case, using Gaussian elimination for the linear algebra step increases the density of the matrix due to the fill-in property. Solving such matrix by F4 [11] and/or XL makes the linear algebra very costly in terms of memory and time. For example, F4 algorithm that is implemented in Magma was not able to solve a dense random system with 32 variables in 32 quadratic equations on a server that has 128GB memory.

On the other hand, sparse linear algebra was successfully used in integer factorization specially the use of block Wiedemann over GF(2) in the number field sieve method to factor the RSA-768 number [12]. This leads to the importance of using the Wiedemann algorithm as a solver instead of Gaussian elimination. Other reasons for using Wiedemann are that when XL extends the system to a certain high degree the resulting system is very sparse and the systems that are initially produced from cryptosystems are very large and sparse, Wiedemann is faster than Gaussian elimination for sparse systems.

In this paper, we represent an algorithm that uses the block Wiedemann algorithm over GF(2) as a solver instead of Gaussian elimination in the XL algorithm, we call it WXL. This idea was used over GF(256) with a scalar version of Wiedemann in [13]. We present an experimental comparison with Magma's implementation of the F4 algorithm, MXL_3 [14], an efficient algorithm for computing Gröbner basis, and WXL. Our experiments are based on random instances of the MQ-problem and some HFE cryptosystems that demonstrate the effect of using such solver. We show that a parallel version of WXL, we call it PWXL, is able to solve an instance of the HFE systems of univariate degree

4352, the same degree as challenge 2, that are generated directly over GF(2) with 34 quadratic equations in 34 variables while Magma's F4 and MXL_3 can not solve any of such systems with more than 31 variables using the same server with 128GB memory.

This paper is organized as follows. In section 2, we give an overview of the Wiedemann algorithm. We then present the WXL algorithm in section 3. In section 4, we introduce experimental results on HFE systems and random systems. A discussion for PWXL is presented in section 5. Finally we conclude the paper in section 6.

2 The Wiedemann Algorithm

In 1986 Wiedemann introduced an algorithm to solve a linear system and compute the determinant of a black box matrix over a finite field [15]. Wiedemann's approach uses the fact that the minimal polynomial of a matrix generates the Krylov sequences of the matrix and their projections. In other words, when a square matrix is repeatedly applied to a vector, the resulting sequence is linear recursive. In 1994 Coppersmith introduced a block version of Wiedemann's algorithm over GF(2) [16]. By using projections of a block of vectors instead of a single vector it is possible to do the parallelization of the matrix times vector products.

The computation of the minimal generating matrix polynomial of the block Wiedemann sequence is an important task. Several algorithms have been introduced to deal with this task. Coppersmith uses a multivariate generalization of the Berlekamp-Massey algorithm, Kaltofen solves a homogeneous block Toeplitz system and Villard proposes the using of Fast Power Hermite-Padé solver (FPHPS) algorithm of Backermann and Labahn [17]. While Emmanuel Thomé [18] presents an algorithm, adapted from the divide-and-conquer approach that yielded the HGCD (half-gcd) algorithm and the PRSDC (polynomial reminder sequence by divide-and-conquer) algorithm.

Consider the system $\mathbf{A}\boldsymbol{x} = \mathbf{b}$ over a finite field GF(q), with $\mathbf{A}$ a non-singular, sparse $n \times n$ matrix. The approach used by Wiedemann is to start from a vector $\mathbf{b}$ and to compute the Krylov sequence $\{uA^ib\}_{i=0}^{2n-1}$, for any row vector $u \in GF(q)^{1\times n}$. This sequence is linearly generated since the set of all theses vectors has dimension $\leq n$, so there exists a non-trivial linear dependency relation between the first $n+1$ vectors of this sequence. Moreover, this sequence provides a minimal recurrence polynomial $F_u^{A,b}(\lambda)$ that can be computed using Berlekamp-Massey algorithm. Wiedemann proved that for random vectors u and b with high probability $F_u^{A,b}(\lambda) = F^A(\lambda)$, where F^A is the minimum polynomial of the matrix $\mathbf{A}$.

Let $F_u^{A,b}(\lambda) = c_0 + c_1\lambda + c_2\lambda^2 + ... + c_d\lambda^d, c_0, c_1, .., c_d \in \text{GF}(q)$, and $c_0 = 1$. The vectors of the Krylov sequence satisfy the linear equation $F^{A,b}(A)b = 0$. Hence $c_0b + c_1Ab + c_2A^2b + ... + c_dA^db = 0$, rearranging we obtain $b = -A(c_1b + c_2Ab + ... + c_dA^{d-1}b)$. If we define $x = -(c_1b + c_2Ab + ... + c_dA^{d-1}b)$ then x is a solution of $\mathbf{A}\boldsymbol{x} = \mathbf{b}$. For the case that $c_0 = 0$, $\mathbf{A}$ is singular, then we have

$c_1Ab + c_2A^2b + ... + c_dA^{d-1}b = 0$ or $A(c_1b + c_2Ab + ... + c_dA^{d-1}b) = 0$. So, we can either find a solution of $\mathbf{A}\boldsymbol{x} = \mathbf{b}$ or a non-trivial element in the kernel ker(A) of A. The block Wiedemann algorithm is presented in algorithm 1.

Algorithm 1 bWiedemann

```
1: Inputs
2:    $A \in GF(2)^{N \times N}$
3: Output
4:    $w \neq 0$ such that $Aw = 0$.
5: Begin
6: Pick up random matrices $X \in GF(2)^{m,N}$, $Y \in GF(2)^{N,n}$. Let $Z = AY$
7: Let $\delta_l = \lceil N/m \rceil$ and $\delta_r = \lfloor N/n \rfloor$. Compute $a_i = XA^iY, i = 0, \dots, \delta_l + \delta_r - 1$.
8: Compute a generating vector polynomial $g(\lambda) = c_l\lambda^l + c_{l+1}\lambda^{l+1} + \cdots + c_d\lambda^d \in GF(2)^n[\lambda]$ of degree at most $\delta_r$ for the sequence $\{XA^iY\}$, where $l \geq 0$, $d \leq \delta_r$, $c_l \neq 0$.
9: Compute $\hat{w} \leftarrow c_lY + c_{l+1}AY + \cdots + c_dA^{d-l}Y$;
   (with high probability $\hat{w} \neq 0$ and $A^{l+1}\hat{w} = 0$)
10: Compute the first $k$ such that $A^k\hat{w} = 0$;
11: If $k \geq 1$ then $w = A^{k-1}\hat{w}$ else $w = 0$
12: End
```

3 WXL: The Wiedemann XL Algorithm

As XL the WXL algorithm starts with linearizing the original system of polynomial equations to construct an equivalent Macaulay matrix of the system at starting degree $D = 2$. This is achieved by replacing each monomial by a new variable. If the constructed Macaulay matrix is not undetermined then we can apply Wiedemann to try to solve, otherwise like XL extends the system to the next higher degree $D + 1$ and we repeat the linearization. In the case that we found a determined or overdetermined matrix then we try to solve it by extracting a square sub-matrix from the extended system at that degree. If there is a solution, we must check whether such a solution for the linearized system is also a solution to the original quadratic system or not. If this solution is satisfiable to the original system then terminate and return the solution, otherwise we may try some other square sub-matrix to be solved again until some limit. After trying up to some limit and there is no solution then extend the system and linearize again. Algorithm 2 describes the WXL algorithm.

The main critical point of WXL is to choose a number of polynomials in order to construct a square Macaulay matrix of the system at a certain degree that can generate as small number of solution as possible for the linearized system. We use a heuristic argument from [13] that if we pick rows at random under the constraint that we have enough equations at each level, then usually we have a linearly independent set. This is exactly what we mean by the function

Algorithm 2 WXL

```
1: Inputs
2:    $P$: set of m quadratic polynomials.
3:    $Limit$: number of maximum trails.
4: Output
5:    $Solution$: A solution of $P$=0.
6: Variables
7:    $\mathcal{M}^{acaulay}$: a matrix whose entries are the coefficients of a system of multivariate
      polynomial equations in graded lex order.
8:    $\mathcal{M}_{sq}^{acaulay}$: a square submatrix.
9:    $\tilde{P}$: set of all polynomials that are included in the system.
10:   $D$: the highest degree of $\tilde{P}$.
11:   $solved$: a flag to indicate whether the system is solved or not.
12:   $attempt$: a counter for the number of trails.
13:   $Wsolution$: set of solutions generated by Wiedemann for the linearized system.
14: Begin
15: Initialization()
    {$\mathcal{M}^{acaulay}$, $Solution$, $Wsolution \leftarrow \varnothing$, $solved \leftarrow$ false, $\tilde{P} \leftarrow P$, $D \leftarrow 2$ }
16: repeat
17:    $\mathcal{M}^{acaulay} \leftarrow$ Linearize($\tilde{P}$)
18:    if $nRows(\mathcal{M}^{acaulay}) \geq nCols(\mathcal{M}^{acaulay})$ then
19:       $attempt \leftarrow 1$
20:       repeat
21:          $\mathcal{M}_{sq}^{acaulay} \leftarrow$ Make_square($\mathcal{M}^{acaulay}$)
22:          $Wsolution \leftarrow$ Wiedemann($\mathcal{M}_{sq}^{acaulay}$)
23:          if $Wsolution \neq \varnothing$ then
24:             ($solved$,$Solution$) $\leftarrow$ Check_solution($P$, $Wsolution$)
25:             if $solved$ then
26:                Return ($Solution$)
27:             end if
28:          end if
29:          $attempt \leftarrow attempt + 1$
30:       until ($attempt \geq Limit$)
31:    end if
32:    $D \leftarrow D + 1$
33:    $\tilde{P} \leftarrow \tilde{P} \cup$ Extend($P$, $D$ )
34: until ($solved$)
35: End
```

Make_square($\mathcal{M}^{acaulay}$) in the WXL algorithm. In all experiments of HFE and dense random systems, WXL always solves using only the first square sub-matrix at a certain degree D. While for some supper sparse random systems, it needs to select more than one such square sub-matrix.

In the WXL algorithm, by Extend(P, D) we mean, multiply each polynomial by a monomial of degree $D-2$. Wiedemann($\mathcal{M}_{sq}^{acaulay}$) applies the block Wiedemann algorithm to a square Macaulay matrix as it is described in algorithm 1.

4 Experimental Results

In this section we present experimental results and compare the performance of WXL with Magma's implementation of F4 and MXL$_3$. We are interested in solving systems of multivariate quadratic polynomial equations when the number of equations is the same as the number of unknowns. We use some instances of dense random systems generated by Courtois [19] and some HFE systems generated by the code of John Baena. All the dense random systems have multiple solutions except for the instance with 24 variables has a unique solution. The central map for the HFE scheme is not necessarily a bijection, therefore we may find more than one solution to such systems.

The complexity of solving systems of dense multivariate quadratic polynomial equations depends on the number of variables in each system. Therefore, the complexity of solving different systems with the same number of variables more or less will be the same. In this framework, the results given in this section are for an instance for each system. All the experiments are done on a Sun X4440 server, with four "Quad-Core AMD Opteron$^{\mathrm{TM}}$Processor 8356" CPUs and 128GB of main memory. Each CPU is running at 2.3 GHz. In these experiments we use only one out of the 16 cores.

We used Magma version (V2.16-1) unless stated. The WXL algorithm we implemented uses block Wiedemann solver written by Thorsten Kleinjung which uses 64 bit word block Wiedemann and returns 64 solutions. It also uses MSLGDC (Matrix Sequences Linear Generator by Divide-and Conquer) algorithm for computing linear generator in subquadratic computation [18]. All experimental data for MXL$_3$ are done by Mohamed Saied Emam Mohamed, the first author of [14].

Tables 1 and 2 show results for HFE systems of univariate degree 4352 that are generated directly over GF(2) and results for dense random systems, respectively. The first column "Sys" denotes the number of variables and the number of equations for each system. The highest degree of the elements of the system can reach is denoted by "D". The used memory in Megabytes and the execution time in seconds is represented by "Mem" and "Time" respectively.

In both tables, we can see that WXL always outperforms F4 in terms of memory, our implementation of WXL is not optimized yet. Therefore Magma's F4 version (V2.16-1) is faster than WXL. Magma's F4 is using a fast, improved , and updated linear algebra. For older versions, table 3 shows that WXL is faster and uses less memory than F4 version (V2.13-10).

Table 1. Performance of WXL among F4 and MXL_3 for HFE(4352, Sys)

Sys	F4			MXL_3			WXL		
	D	Mem	$Time$	D	Mem	$Time$	D	Mem	$Time$
24	6	3557	274	6	390	352	6	436	788
25	6	7561	310	6	607	800	6	524	1321
26	6	11910	629	6	1208	1463	6	814	2324
27	6	17722	1293	6	2437	3880	6	920	4074
28	6	22086	2976	6	4844	10735	6	1057	7348
29	6	32370	6811	6	9626	19802	6	1619	11925
30	6	95086	61257	6	15050	33818	7	9167	394800
31	6	115,875	98,797	6	23,168	94,280	7	19,909	439,925

Table 2. Performance of WXL among F4 and MXL_3 for dense random systems

Sys	F4			MXL_3			WXL		
	D	Mem	$Time$	D	Mem	$Time$	D	Mem	$Time$
24	6	3558	235	6	392	341	6	514	800
25	6	7561	457	6	698	704	6	578	1290
26	6	11926	957	6	1207	1429	6	749	2413
27	6	17717	2187	6	2315	2853	6	1052	4516
28	6	22088	4096	6	4836	7982	6	1403	7159
29	6	32392	7701	6	9375	18796	6	1797	11312

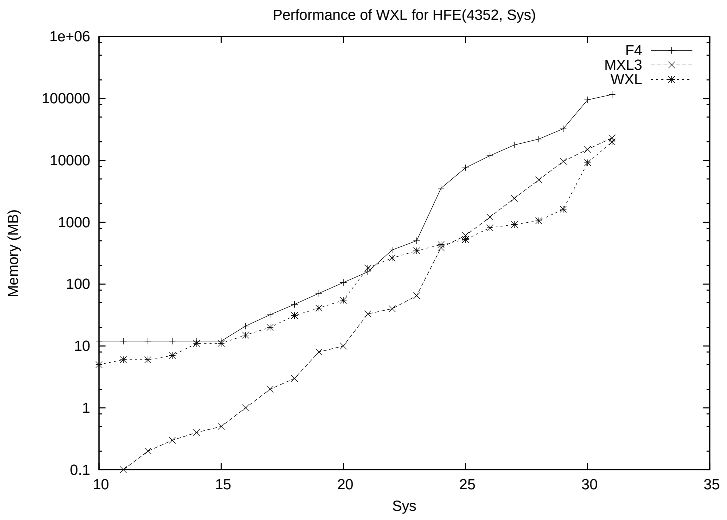


Fig. 1. Performance of WXL among F4 and MXL_3 for HFE(4352, 10-31)

Table 3. Performance of WXL versus $F4_{v2.13-10}$

Sys	$F4_{v2.13-10}$			WXL		
	D	Mem	$Time$	D	Mem	$Time$
24	6	3071	855	6	514	800
25	6	5128	1341	6	578	1290
26	6	8431	3325	6	749	2413
27	6	13312	6431	6	1052	4516
28	6	20433	13810	6	1403	7159
29	6	30044	25631	6	1797	11312

Table 4. Matrix dimensions for MXL_3, F4, and WXL

Sys	MXL_3	F4	WXL
24	57183×57171	207150×78637	190051×190051
25	66631×76414	248795×109046	245506×245506
26	88513×102246	298673×148885	313912×313912
27	123938×140344	354294×198007	397594×397594
28	201636×197051	420773×261160	499178×499178
29	279288×281192	499258×340290	621616×621616

WXL also outperforms MXL_3 in terms of memory for systems with number of variables greater than 24 and for systems with 28-29 variables, WXL is faster and consumes less memory than MXL_3. Starting from 30 variables WXL is worse in terms of time against MXL_3 this because WXL solves at degree 7 while MXL_3 solves at degree 6. Figure 1 shows a comparison among F4, MXL_3, and WXL in terms of memory for HFE systems that have a number of variables 10-31.

In table 4, we compare the matrix dimensions with MXL_3, F4, and WXL. It is obvious that WXL has the biggest matrix dimensions because WXL did not have an optimized selection strategy for extending polynomials which is the key for MXL_3 and F4.

We realized that MXL_3 is better than WXL in both memory and time for systems that have number of variables less than 25. The main reason for that is MXL_3 has an optimized selection strategy that makes the systems solved with a very small matrix dimensions compared to WXL. While WXL is better only in memory but not in time for systems that have number of variables greater than 24. For systems that have a number of variables 28 and 29, WXL is better in both time and memory. MXL_3 uses mutants that make some systems solved at lower degree than WXL. This is the reason that WXL takes a lot of time in the instances that have 30 and 31 variables.

5 PWXL: A Parallelization of WXL

In this section we discuss the parallelized version of the WXL algorithm and the promising results that can be obtained from it. Obviously, the last version of

Magma's implementation of F4 is faster than WXL. One way to improve WXL is to use more than one processor based on the advantage that the Wiedemann algorithm is applicable to be parallelized while F4 is not easy to be parallelized.

There are several implementations of the block Wiedemann algorithm. We test WLSS2 [20] and Kleinjung's code that is used to factor RSA-768 number [12]. The latter is more efficient than WLSS2. Therefore we focus only on it to be the solver for our PWXL algorithm. Kleinjung's code is implemented in C and parallelized by MPI (Message Passing Interface).

It consists of five separated programs that communicate by means of files. Each program is corresponding to a step in algorithm 1. We add to these five programs another two. The first one is for preprocessing that is responsible for extending the original system up to a certain degree, select a square matrix from the extended system, then convert the square matrix to the proper format that is accepted by the first program in Kleinjung's code. The second program is responsible for checking the solution that is generated from Wiedemann and returns it such that it is also a solution for the original system otherwise it returns a no solution message.

A big advantage of PWXL is the fact that the linear systems can be generated in a distributed fashion. Since we know what the initial system is and what the effect of the multiplication by monomials is, we can generate the parts of the system on the respective nodes without the need to store the full linear system in one place at any time.

The experimental server for these experiments is a SUN X4440, with four "Six-Core AMD Opteron^{TM}Processor 8435" CPUs running at 2.6 GHz each, 24 Cores in total and 64 GB System memory. The reasons for not to use the same server as in section 4 are: there is no more free cores in this server and we have only a license for Magma at it.

The run time of Wiedemann depends on the number P of processors, but in a more complex way. Basically there are local computations and there is a communication between processors. The later depends on the topology of the network; a torus topology of $P = P_1 \times P_2$ processors with $P_1 \approx P_2$ seems to be a good choice.

In table 5, we compare the performance of the PWXL algorithm in time using 1, 2, 4, 8, 16 processors to HFE systems with univariate degree 4352 for 24-33 equations in 24-33 variables. The time measured in seconds except for bigger systems it is in hours (H) or days (D).

In [14], the authors noticed that when MXL$_3$ and F4 tried to solve a 32 variable system, both solvers were unable to extend the system to degree 7 because of memory. While PWXL solves systems starting from number of variable equal to 30 at degree 7. Also, PWXL can successfully solve an instance of HFE(4352, 34) in 13.17 days using 16 processors.

Table 5. Experimental Results for PWXL

Sys	1P	2P	4P	8P	16P
24	730	408	214	143	96
25	1295	762	438	288	158
26	2398	1306	721	553	245
27	4456	2422	1397	751	464
28	7686	3921	2874	1602	889
29	13790	8004	4325	2715	1945
30	4.62D	3.26D	2.45D	22.30H	15.4H
31	12.33D	6.41D	4.07D	2.88D	22.03H
32	16.70D	11.72D	6.13D	4.93D	1.73D
33	32.79D	20.27D	11.46D	7.93D	3.18D

6 Conclusion and Future Work

In this paper, we represent the WXL algorithm for solving multivariate quadratic polynomial systems over GF(2) that is based on the block Wiedemann algorithm instead of Gaussian elimination as a solver. Experimentally, WXL is better than Magma's F4, which is known to be the best available implementation of F4, in terms of memory for HFE and dense random systems. Moreover, by using PWXL, a parallelized version of WXL, we are able to solve systems with higher number of variables that aren't solved by other solvers.

We are interested in solving instances of random and HFE systems of univariate degree 4352 which is the same degree of the HFE challenge 2. From experimental point of view, we can conclude that HFE systems of univariate degree 4352 over GF(2) have the same complexity of random systems within the number of variables from 10 to 34.

We plan to use more processors in order to push PWXL to solve systems with more number of variables taking into account the number of processors versus the number of variables. We also intend to use some ideas from structured Gaussian elimination to minimize the matrix dimensions and at the same time keep the sparsity not changed as possible.

The PWXL implementation is applicable only for systems that can be solved at a degree where the number of polynomials is greater than or equal to the number of monomials. So we intend to use the concept of mutant to solve such shortage. The mixing between MXL_3 and PWXL is a promising tool that can improve the field of algebraic cryptanalysis.

Acknowledgment

We would like to thank John Baena for supporting us with his code of generating HFE systems.

References

1. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman and Co., New York, NY, USA (1990)
2. Courtois, N.T., Pieprzyk, J.: Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In: ASIACRYPT 2002. Volume 2501 of Lecture Notes in Computer Science., Queenstown, New Zealand, Springer-Verlag, Berlin (2002) 267–287
3. Courtois, N.T.: Higher Order Correlation Attacks, XL Algorithm and Cryptanalysis of Toyocrypt. In: proceding of 5th International Conference on Information Security and Cryptology (ICISC). Volume 2587 of Lecture Notes in Computer Science., Seoul, Korea, Springer-Verlag (2002) 182–199
4. Armknecht, F., Krause, M.: Algebraic Attacks on Combiners with Memory. In: CRYPTO. Volume 2729 of Lecture Notes in Computer Science., Santa Barbara, California, USA, Springer (2003) 162–175
5. Faugère, J.C., Joux, A.: Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases. In: Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference Proceedings, Springer (2003) 44–60
6. Deursen, T., Radomirović, S.: Algebraic Attacks on RFID Protocols. In: WISTP '09: Proceedings of the 3rd IFIP WG 11.2 International Workshop on Information Security Theory and Practice. Smart Devices, Pervasive Systems, and Ubiquitous Networks, Brussels, Belgium, Springer-Verlag (2009) 38–51
7. Patarin, J.: Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of Asymmetric Algorithms. In: Proceeding of International Conference on the Theory and Application of Cryptographic Techniques Advances in Cryptology- Eurocrypt. Volume 1070 of Lecture Notes in Computer Science., Saragossa, Spain, Springer (1996) 33–48
8. Courtois, N.T.: Algebraic Attacks over GF(2^k), Application to HFE Challenge 2 and Sflash-v2. In: Public Key Cryptography (PKC 2004), 7th International Workshop on Theory and Practice in Public Key Cryptography. Volume 2947 of Lecture Notes in Computer Science., Springer (2004) 201–217
9. Courtois, N.T., Klimov, A., Patarin, J., Shamir, A.: Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In: EUROCRYPT 2000. Volume 1807 of Lecture Notes in Computer Science., Bruges, Belgium, Springer (2000) 392–407
10. Ars, G., Faugère, J.C., Imai, H., Kawazoe, M., Sugita, M.: Comparison between XL and Gröbner Basis Algorithms. In: ASIACRYPT 2004. Volume 3329 of Lecture Notes in Computer Science., Jeju Island, Korea, Springer Berlin / Heidelberg (2004) 338–353
11. Faugére, J.C.: A New Efficient Algorithm for Computing Gröbner Bases (F4). Pure and Applied Algebra **139** (1999) 61–88
12. Kleinjung, T., Aoki, K., Franke, J., Lenstra, A., Thom, E., Bos, J., Gaudry, P., Kruppa, A., Montgomery, P., Osvik, D.A., te Riele, H., Timofeev, A., Zimmermann, P.: Factorization of a 768-bit RSA Modulus. Cryptology ePrint Archive, Report 2010/006 (2010) `http://eprint.iacr.org/`.
13. Yang, B.Y., Chen, O.C.H., Bernstein, D.J., Chen, J.M.: Analysis of QUAD. In: Fast Software Encryption: 14th International Workshop, (FSE 2007). Lecture Notes in Computer Science, Luxembourg, Luxembourg, Springer-Verlag (2007) 290–308

14. Mohamed, M.S.E., Cabarcas, D., Ding, J., Buchmann, J., Bulygin, S.: MXL3: An Efficient Algorithm for Computing Gröbner Bases of Zero-dimensional Ideals. In: Proceedings of The 12th international Conference on Information Security and Cryptology, (ICISC 2009). Lecture Notes in Computer Science, Springer-Verlag, Berlin (2009)
15. Wiedemann, D.H.: Solving Sparse Linear Equations over Finite Fields. IEEE Trans. Inf. Theor. **32** (1986) 54–62
16. Coppersmith, D.: Solving Homogeneous Linear Equations Over GF(2) via Block Wiedemann Algorithm. Math. Comput. **62** (1994) 333–350
17. Turner, W.J.: A Block Wiedemann Rank Algorithm. In: ISSAC '06: Proceedings of the 2006 international symposium on Symbolic and algebraic computation, New York, NY, USA, ACM (2006) 332–339
18. Thomé, E.: Subquadratic Computation of Vector Generating Polynomials and Improvement of the Block Wiedemann Algorithm. J. Symb. Comput. **33** (2002) 757–775
19. Courtois, N.T.: Experimental Algebraic Cryptanalysis of Block Ciphers. http://www.cryptosystem.net/aes/toyciphers.html (2007)
20. Kaltofen, E., Lobo, A.: Distributed Matrix-Free Solution of Large Sparse Linear Systems over Finite Fields. Algorithmica **24** (1997) 331–348

Analysis of the MQQ Public Key Cryptosystem

Rune Ødegård[1] ⋆, Ludovic Perret[2], Jean-Charles Faugère[2], and Danilo Gligoroski[3]

[1] Centre for Quantifiable Quality of Service in Communication Systems at the Norwegian University of Science and Technology in Trondheim, Norway.
rune.odegard@q2s.ntnu.no

[2] Institut National de Recherche en Informatique et en Automatique, Solving ALgebraic Systems and Applications Project
Laboratoire d'Informatique de Paris 6, Universite Pierre et Marie Curie, France
ludovic.perret@lip6.fr
Jean-Charles.Faugere@grobner.org

[3] Department of Telematics at the Norwegian University of Science and Technology in Trondheim, Norway
danilog@item.ntnu.no

Abstract. MQQ is a multivariate cryptosystem based on multivariate quadratic quasigroups and the Dobbertin transformation [18]. The cryptosystem was broken both by Gröbner bases computation and MutantXL [27]. The complexity of Gröbner bases computation is exponential in the degree of regularity, which is the maximum degree of polynomials occurring during the computation. The authors of [27] observed that the degree of regularity for solving the MQQ system is bounded from above by a small constant. In this paper we go one step further in the analysis of MQQ. We explain why the degree of regularity for the MQQ system is bounded. The main result of this paper is how the complexity of solving the MQQ system is the minimum complexity of solving just one quasigroup block and solving the Dobbertin transformation. Furthermore, we show that the degree of regularity for solving the Dobbertin transformation is bounded from above by the same constant as the bound on the MQQ system. We then investigate the strength of a tweaked MQQ system where the input to the Dobbertin transformation is replaced with random linear equations. We find that the degree of regularity for this tweaked system varies both in the size of the quasigroups and the number of variables. We conclude that if a suitable replacement for the Dobbertin transformation is found, MQQ can possibly be made strong enough to resist pure Gröbner attack for correct choices of quasigroups size and number of variables.

Keywords: multivariate cryptography, Gröbner bases , public-key, multivariate quadratic quasigroups, algebraic cryptanalysis

⋆ Rune Steinsmo Ødegård was visiting the SALSA team at LIP6 during the research of this paper.

1 Introduction

Multivariate cryptography comprises all the cryptographic schemes that use multivariate polynomials. At first glance, many aspects of such systems are tempting for cryptographers. Basing schemes on the hard problem of solving a system of multivariate equations is very appealing for multiple reasons. Most importantly, generic algorithms to solve this problem are exponential in the worst case, and solving random system of algebraic equations is also known to be difficult (i.e. exponential) in the average case. Moreover, no quantum algorithm allowing to solve non linear equations exists. Finally, multivariate schemes usually require computations with rather small integers leading to rather efficient smart-card implementations (see for example [7]).

The use of polynomial systems in cryptography dates back to the mid eighties with the design of Matsumoto and Imai [26], later followed by numerous other proposals. Two excellent surveys on the current state of proposals for multivariate asymettric cryptosystems has been made by Wolf and Prenel [34] and Billet and Ding [6]. Basicly the current proposals can be classified into four main categories, some of which combine features from several categories: Matsumoto-Imai like schemes [29,31], Oil and Vinegar like schemes [30,21], Stepwise Triangular Schemes [32,19] and Polly Cracker Schemes [12]. In addition Gligoroski et al. has proposed a fifth class of trapdoor functions based on multivariate quadratic quasigroups [18].

Unfortunately, it appears that most multivariate public-key schemes suffer from obvious to less obvious weaknesses. This is evident in [6] where a nice overview of the cryptanalysis techniques in multivariate asymmetric cryptography is given. Some attacks are specific attacks which focus on one particular variation and breaks it due to specific properties. One example of this is the attack of Kipnis and Shamir against Oil and Vinegar [22]. However most attacks use general purpose algorithms that solve multivariate system of equations. As mentioned, algorithms for solving random system of equations are known to be exponential in the avarage case. However in the case of multivariate public-key schemes the designer has to embed some kind of trapdoor function to enable efficient decryption and signing. To acheive this the public-key equations are constructed from a highly structured system of equations. Although the structure is hidden, it can be exploited for instance via differential or Gröbner based techniques.

Gröbner basis [9] is a well established and general method for solving polynomial systems of equations. The complexity of Gröbner bases computation is exponential in the degree of regularity, which is the maximum degree of polynomials occurring during the computation [4]. The first published attack on multivariate public-key cryptosystems using Gröbner basis is the attack by Paterin on the Matsumoto-Imai scheme [28]. In the paper Patarin explains exactly why Gröbner bases is able solve the system. The key remark is that there exists bilinear equations relating the input and the output of the system [6]. This low degree relation between the input and the output means that only polynomials of low degree will occur during the computation of Gröbner bases. As a result the complexity of solving the system is bounded in this low degree.

Another multivariate cryptosystem that has fallen short for Gröbner bases cryptanalysis is the MQQ public key block cipher [18]. The cipher was solved both by Gröbner bases and MutantXL independently in [27]. However [27] did not theoretically explain why the algebraic systems of MQQ are easy to solve in practice. In this paper we explain exactly why the MQQ cryptosystem is susceptible to algebraic cryptanalysis. This is of course interesting from a cryptanalysis perspective, but also from a design perspective. If we want to construct strong multivariate cryptographic schemes we must understand why the weak schemes have been broken.

1.1 Organisation of the paper

This paper is organized as follows. In Section 2 we give an introduction to multivariate quadratic quasigroups. After that we describe the MQQ public key cryptosystem. In Section 3 we give a short introduction to the theory of Gröbner basis and recall the theoretical complexity of computing such bases. The complexity of computing Gröbner bases is exponential in the degree of regularity, which is the maximal degree of the polynomials occurring during computation. In Section 4 we show that the degree of regularity of MQQ systems is bounded from above by a small constant. We then explain this behavior thanks to the shape of the inner system. In Section 5 we further elaborate on the weaknesses of the MQQ system, and investigate if some tweaks can make the system stronger. Finally, Section 6 concludes the paper.

2 Description of MQQ public key cryptosystem

In this section we give a description of the multivariate quadratic quasigroup public key cryptosystem [18]. The system is based on previous work by Gligoroski and Markovski who introduced the use of quasigroup string processing in cryptography [24,25].

2.1 Multivariate quadratic quasigroups

We first introduce the key building block namely multivariate quadratic quasigroups. For a detailed introduction to quasigroups in general we refer the interested reader to [33].

Definition 1 *A quasigroup is a set Q together with a binary operation $*$ such that for all $a, b \in Q$ the equations $\ell * a = b$ and $a * r = b$ have unique solutions ℓ and r in Q. A quasigroup is said to be of order n if there are n elements in the set Q.*

Let $(Q, *)$ be a quasigroup of order 2^d, and β be a bijection from the quasigroup to the set of binary strings of length d, i.e

$$\begin{aligned} \beta : Q \to & \quad \mathbb{Z}_2^d \\ a \mapsto & (x_1, \ldots, x_d) \end{aligned} \tag{1}$$

Given such a bijection we can naturally define a vector valued Boolean function

$$\begin{aligned} *_{vv} : \quad \mathbb{Z}_2^d \times \mathbb{Z}_2^d & \to \quad \mathbb{Z}_2^d \\ (\beta(a), \beta(b)) & \mapsto \beta(a * b) \end{aligned} \tag{2}$$

Now let $\beta(a * b) = (x_1, \ldots, x_d) *_{vv} (x_{d+1}, \ldots, x_{2d}) = (z_1, \ldots, z_d)$. Note that each z_i can be regarded as a $2d$-ary Boolean function $z_i = f_i(x_1, \ldots, x_{2d})$, where each $f_i : \mathbb{Z}_2^d \to \mathbb{Z}_2$ is determined by $*$. This gives us the following lemma [18].

Lemma 1 *For every quasigroup $(Q, *)$ of order 2^d and for each bijection $\beta : Q \to \mathbb{Z}_2^d$ there is a unique vector valued Boolean function $*_{vv}$ and d uniquely determined $2d$-ary Boolean functions $f_1, f_2, \ldots, f_d$ such that for each $a, b, c \in Q$:*

$$\begin{gathered} a * b = c \\ \Updownarrow \\ (x_1, \ldots, x_d) *_{vv} (x_{d+1}, \ldots, x_{2d}) = (f_1(x_1, \ldots, x_{2d}), \ldots, f_d(x_1, \ldots, x_{2d})). \end{gathered} \tag{3}$$

This leads to the following definition for multivariate quadratic quasigroups.

Definition 2 *([18]) Let $(Q, *)$ be a quasigroup of order 2^d, and let $f_1, \ldots, f_d$ be the uniquely determined Boolean functions under some bijection β. We say that the quasigroup is multivariate quadratic quasigroup (MQQ) of type $Quad_{d-k}Lin_k$ (under β) if exactly $d - k$ of the corresponding polynomials f_i are of degree 2 and k of them are of degree 1, where $0 \leq k \leq d$.*

Gligoroski et al. mention [18] that quadratic terms might cancel each other. By this we mean that some linear transformation of $(f_i)_{1 \leq i \leq n}$ might result in polynomials where the number of linear polynomials is larger than k, while the number of quadratic polynomials is less than $d - k$. Later Chen et al. [10] have shown that this is more common than previously expected. In their paper they generalizes the definition of MQQ above to a family which is invariant by linear transformations in $\mathbb{Z}_2[x_1, \ldots, x_{2d}]$.

Definition 3 *Let $(Q, *)$ be a quasigroup of order 2^d, and let $f_1, \ldots, f_d$ be the unique Boolean functions under some bijection β. We say that the quasigroup is a multivariate quadratic quasigroup (MQQ) of strict type $Quad_{d-k}Lin_k$ (under β), denoted by $Quad^s_{d-k}Lin^s_k$, if there are at most $d - k$ quadratic polynomials in $(f_i)_{1 \leq i \leq d}$ whose linear combination do not result in a linear form.*

Chen et al. also improves Theorem 2 from [18] which gives a sufficient condition for a quasigroup to be MQQ. We restate this theorem below.

Theorem 1 *Let $\mathbf{A}_1 = [f_{ij}]_{d\times d}$ and $\mathbf{A}_2 = [g_{ij}]_{d\times d}$ be two $d \times d$ matrices of linear Boolean expressions with respect to $x_1, \dots, x_d$ and $x_{d+1}, \dots, x_{2d}$ respectively. Let $\mathbf{c}$ be a binary column vector of d elements. If $det(\mathbf{A}_1) = det(\mathbf{A}_2) = 1$ and*

$$\mathbf{A}_1 \cdot (x_{d+1}, \dots, x_{2d})^T + (x_1, \dots, x_d)^T = \mathbf{A}_2 \cdot (x_1, \dots, x_d)^T + (x_{d+1}, \dots, x_{2s})^T, \quad (4)$$

then the vector valued Boolean operation

$$(x_1, \dots, x_d) *_{vv} (x_{d+1}, \dots, x_{2d}) = \mathbf{B}_1\mathbf{A}_1 \cdot (x_{d+1}, \dots, x_{2d})^T + \mathbf{B}_2 \cdot (x_1, \dots, x_d)^T + \mathbf{c} \quad (5)$$

defines a quasigroup $(Q, *)$ of order 2^d which is MQQ for any non-singular Boolean matrices $\mathbf{B}_1$ and $\mathbf{B}_2$

In addition Chen et al. proved [10] that no MQQ as in Theorem 1 can be of strict type $\text{Quad}_d^s\text{Lin}_0^s$. This result uncovered a possible weakness in [18] since the proposed scheme is using 6 quasigroups of type $\text{Quad}_5\text{Lin}_0$.

Notice that the vector valued Boolean function defining the MQQ in Theorem 1 have no terms of the form $x_i x_j$ with $i, j \leq d$ or $i, j > d$. This means that if we set the first or the last half of the variables to a constant, we end up with only linear terms in the MQQ. It is still an open question if there exists MQQ that are not as in Theorem 1.

The MQQs used in this paper has been produced using the algorithm provided in Appendix A. The algorithm is based on the paper [10], and produces MQQs that are more suitable for encryption since they are guaranteed to be of strict type $\text{Quad}_{d-k}^s\text{Lin}_k^s$.

2.2 Dobbertin bijection

In addition to MQQs, the public key cryptosystem [18] also uses a bijection introduced by Dobbertin in [13]. Dobbertin proved that the following function, in addition to being multivariate quadratic, is a bijection in $\mathbb{Z}_{2^{2r+1}}$.

$$\begin{array}{rcl} D_r : \mathbb{Z}_{2^{2r+1}} & \rightarrow & \mathbb{Z}_{2^{2r+1}} \\ x & \mapsto & x^{2^{r+1}+1} + x^3 + x \end{array} \quad (6)$$

2.3 Public key cryptosystem based on MQQ

We are now ready to describe the public key cryptosystem presented by Gligoroski et al. in [18]. Let $N = nd$ be the desired number of variables $(x_1, \dots, x_N)$, and let $\{*_{vv}^1, \dots, *_{vv}^k\}$ be a collection of MQQs of size 2^d represented as $2d$-ary vector valued Boolean functions. The public key is constructed as follows.

Algorithm *MQQ public key construction.*
1. Set $\mathbf{X} = [x_1, \dots, x_N]^T$. Randomly generate a $N \times N$ non-singular Boolean matrix $\mathbf{S}$, and compute $\mathbf{X} \leftarrow \mathbf{S} \cdot \mathbf{X}$.
2. Construct an n-tuple $I = \{i_1, \dots, i_n\}$, where $i_j \in \{1, \dots, k\}$. The tuple I will decide which MQQ, $*_{vv}^{i_j}$, to use at each point of the quasigroup transformation.

3. Represent $\mathbf{X}$ as a collection of vectors of length d, $\mathbf{X} = [X_1, \ldots, X_n]^T$. Compute $\mathbf{Y} = [Y_1, \ldots, Y_n]^T$ where $Y_1 = X_1, Y_2 = X_1 *_{vv}^{i_1} X_2$, and $Y_j = X_j *_{vv}^{i_j} X_{j+1}$ for $j = 1, \ldots, n$.
4. Set $\mathbf{Z}$ to be the vector of all the linear terms of $Y_1, \ldots, Y_j$. Here Y_1 will be all linear terms, while each Y_j has between 1 and k linear terms depending on the type $\text{Quad}_{d-k}^s \text{Lin}_k^s$ of MQQ used. Transform $\mathbf{Z}$ with one or more Dobbertin bijections of appropriate size. For example if $\mathbf{Z}$ is of size 27 we can use one Dobbertin bijection of dimension 27, three of dimension 9, or any other combination adding up to 27. $\mathbf{W} \leftarrow \text{Dob}(\mathbf{Z})$.
5. Replace the linear terms of $\mathbf{Y} = [Y_1, \ldots, Y_n]^T$ with the terms in $\mathbf{W}$. Randomly generate a $N \times N$ non-singular Boolean matrix $\mathbf{T}$, and compute $\mathbf{Y} \leftarrow \mathbf{T} \cdot \mathbf{Y}$
6. **return** the public key $\mathbf{Y}$. The private key is $\mathbf{S}, \mathbf{T}, \{*_{vv}^1, \ldots, *_{vv}^k\}$ and I.

3 Gröbner basis cryptanalysis

In recent years Gröbner bases has been used as a tool to mount efficient algebraic cryptanalysis [6] In particular, Gröbner bases has been used to attack MQQ [27]. In this paper we go one step further by explaining the weakness found in [27]. In addition we investigate the possibility of constructing stronger MQQ systems.

3.1 Short introduction to Gröbner bases

This section introduces the concept of Gröbner bases. We refer to [11] for basic definitions, and a more detailed description of the concepts.

Let $\mathbb{K}$ be a field and $\mathbb{K}[x_1, \ldots, x_n]$ the polynomial ring over $\mathbb{K}$ in the variables $x_1, \ldots, x_n$. Recall that a *monomial* in a collection of variables is a product $x^\alpha = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ where $\alpha_i \geq 0$. Let $>$ be an admissible *monomial order* on $k[x_1, \ldots, x_n]$. The most common example of monomial order is the *lexicographical order* where $x^\alpha > x^\beta$ if in the difference $\alpha - \beta \in \mathbb{Z}^n$ the leftmost nonzero entry is positive. Another frequently encountered order is the *graded reverse lexicographical* order where $x^\alpha > x^\beta$ iff $\sum_i \alpha_i > \sum_i \beta_i$ or $\sum_i \alpha_i = \sum_i \beta_i$ and in the difference $\alpha - \beta \in \mathbb{Z}^n$ the rightmost nonzero entry is negative. For different orders Gröbner bases has specific theoretical property and different practical behaviors. Given a monomial order $>$ the *leading term* of a polynomial $f = \sum_\alpha c_\alpha x^\alpha$, denoted $LT_>(f)$, is the product $c_\alpha x^\alpha$ where x^α is the largest monomial appearing in f in the ordering $>$.

Definition 4 *([11]) Fix a monomial order $>$ on $k[x_1, \ldots, x_n]$, and let $I \subset k[x_1, \ldots, x_n]$ be an ideal. A Gröbner basis for I (with respect to $>$) is a finite collection of polynomials $G = \{g_1, \ldots, g_t\} \subset I$ with the property that for every nonzero $f \in I$, $LT_>(f)$ is divisible by $LT_>(g_i)$ for some i.*

Let

$$f_1(x_1, \ldots, x_n) = \cdots = f_m(x_1, \ldots, x_n) = 0 \tag{7}$$

by a system of m polynomials in n unknowns over the field $\mathbb{K}$.The set of solutions in $\mathbb{K}$, which is the algebraic variety, is defined as

$$V = \{(z_1, \ldots, z_n) \in k | f_i(z_1, \ldots, z_n) = 0 \forall 1 \le i \le n\} \tag{8}$$

In our case we are interested in the solutions of the MQQ system, which is defined over $\mathbb{Z}_2$.

Proposition 1 *([16]) Let G be a Gröbner bases of $[f_1, \ldots, f_m, x_1^2 - x_1, \ldots, x_n^2 - x_n]$. It holds that:*

1. *$V = \emptyset$ (no solution) iff $G = [1]$.*
2. *V has exactly one solution iff $G = [x_1 - a_1, \ldots, x_n - a_n]$ where $a_i \in \mathbb{Z}_2$. Then $(a_1, \ldots, a_n)$ is the solution in $\mathbb{Z}_2$ of the algebraic system.*

From the proposition we learn that in order to solve a system over $\mathbb{Z}_2$ we should add the field equations $x_i^2 = x_i$ for $i = 1, \ldots, n$. This means that we have to compute a Gröbner bases of $m + n$ polynomials and n variables. This is quite helpful, since the more equations you have, the more able you are to compute Gröbner bases [16].

3.2 Complexity of computing Gröbner bases

Historically the concept of Gröbner bases, together with an algorithm for computing them, was introduced by Bruno Buchberger in his PhD-thesis [9]. Buchberger's algorithm is implemented in many computer algebra systems. However, in the last decade, more efficient algorithms for computing Gröbner bases have been proposed. Most notable are Jean-Charles Faugère's F_4[14] and F_5 [15] algorithms. In this paper we have used the magma [23] 2.16-1 implementation of the F_4 algorithm on a 4 core Intel Xeon 2.93GHz computer with 128GB of memory.

The complexity of computing a Gröbner basis of an ideal I depends on the maximal degree of the polynomials appearing during the computation. This degree, called *degree of regularity*, is the key parameter for understanding the complexity of Gröbner basis computations [4]. Indeed, the complexity of the computation is polynomial in the degree of regularity D_{reg}, more precisely the complexity is:

$$\mathcal{O}(N^{\omega D_{\text{reg}}}), \tag{9}$$

which basically correspond to the complexity of reducing a matrix of size $N^{D_{\text{reg}}}$. Here $2 < \omega \le 3$ is the "linear algebra constant", and N the number of variables of the system. Note that D_{reg} is also a function of N, where the relation between D_{reg} and N depends on the specific system of equations. This relation is well understood for regular (and semi-regular) systems of equations [1,4,2,5]. On the contrary, as soon as the system has some kind of structure, this degree is much more difficult to predict. In some particular cases, it is however possible to bound the degree of regularity (see the works done on HFE [16,20]). But it is a hard task in general.

Note that the degree of regularity is related to the ideal $I = \langle f_1, \ldots, f_n \rangle$ and not the equations $f_1, \ldots, f_n$ themselves. This means given any non-singular matrix S and linear transformation $[f'_1, \ldots, f'_n]^T = S \cdot [f_1, \ldots, f_n]^T$, the degree of regularity for solving equations $f'_1, \ldots, f'_n$ with Gröbner bases is the same as for equations $f'_1, \ldots, f'_n$ since $\langle f'_1, \ldots, f'_n \rangle = \langle f_1, \ldots, f_n \rangle$. More generally, we can assume that this degree is invariant for a (invertible) linear change of variables, and (invertible) combination of the polynomials. These are exactly the transformations performed to mask the MQQ structure.

4 Why MQQ is susceptible to algebraic cryptanalysis

In [27] MQQ systems with up to 160 variables was broken using both the MutantXL and the F_4 algorithm independently. The most important remark by [27] is that the degree of regularity is bounded from above by 3. This is much lower than a random system of quadratic equations where the degree of regularity increases linearly in the number of equations N. Indeed, for a random system it holds that D_{reg} is asymptotically equivalent to $\frac{N}{11.114}$ [2]. The authors of [27] observed that the low degree for MQQ is due to the occurrence of many new low degree relations during the computation of the Gröbner basis. Here, we go one step further in the analysis. We explain precisely why low-degree relations appear. This is due to the very structure of the MQQ system as we explain in detail in Section 4.2. First, we show that we observe the same upper bound on the degree of regularity using the improved quasigroups described in Section 2.1.

4.1 Experimental results on MQQ

To test how the complexity of Gröbner bases computation of MQQ public key systems is related to the number of variables, we constructed MQQ systems of size $30, 60, 120, 180$ following the procedure described in Section 2.3. In this construction we used 17 MQQs of strict type $\text{Quad}_8^s\text{Lin}_2^s$ and Dobbertin bijections over different extension fields of dimension 7 and 9 respectively. The results of this test are presented in Table 1. From the table we see that the degree of regularity does not increase with the number of variables, but remains constant at 3.

Table 1. Results for MQQ-(30,60,120,180). Computed with magma 2.16-1's implementation of the F_4 algorithm on a 4 processor Intel Xeon 2.93GHz computer with 128GB of memory.

Variables	D_{reg}	Solving Time (s)	Memory (b)
30	3	0,06	15,50
60	3	1,69	156,47
120	3	379,27	4662,00
180	3	4136,31	28630,00

Once again, this is not the behaviour of a random system of equations for which the degree of regularity is asymptotically linear in the number of variables. We explain the reason of such difference in the next section.

4.2 Shape of the MQQ system

This non-random behavior can be explained by considering the shape of the "unmasked" MQQ system. By unmasked we mean the MQQ system without the linear transformation S and T. The maximal degree of the polynomials occurring in the computation of a Gröbner basis is invariant under the linear transformation S and T as explained in Section 3.2. In Figure 1 we show what variables appear in each equation for an unmasked MQQ system of 60 variables. The staircase shape comes from the cascading use of quasigroups, while the three blocks of equations at the bottom are from Dobbertin bijection of size 7. A

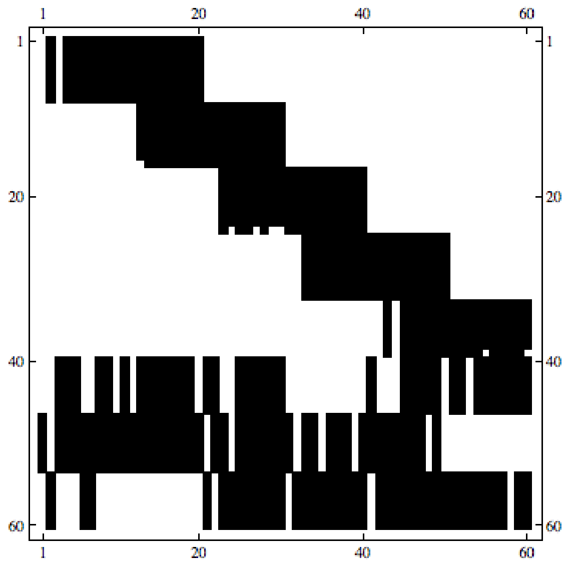


Fig. 1. Shape of 60 variable MQQ public key system without the use of S and T transformation. Black means that the corresponding variables is used in the equation. The system was constructed with 4 MQQs of type $\text{Quad}_8^s\text{Lin}_2^s$, one MQQ of type $\text{Quad}_7^s\text{Lin}_3^s$, and 3 Dobbertin bijections defined over 3 different extension fields of dimension 7.

random multivariate system would use all 60 variables in all equations. For the MQQ system in this example only $\frac{1}{3}$ of the variables are used in each quasigroup and about $\frac{2}{3}$ is used in each block of Dobbertin transformation.

Now assume the Gröbner basis algorithm somewhere during the calculation has found the solution for one of the quasigroup blocks $Y_j = X_j *_{vv}^{i_j} X_{j+1}$. Due to the cascading structure of the MQQ system the variables of X_j are used in the block $Y_{j-1} = X_{j-1} *_{vv}^{i_{j-1}} X_j$ and the variables of X_{j+1} are used in the block. $Y_{j+1} = X_{j+1} *_{vv}^{i_{j+1}} X_{j+2}$. Remember from Section 2.1 that if we set the first or the last half of the variables of an MQQ to constant all equations becomes linear. This means that if we have solved the block Y_j, the equations of the blocks Y_{j-1} and Y_{j+1} becomes linear. The blocks Y_{j-1} and Y_{j+1} can then be solved easily. This gives us solution for the variables X_{j-1} and X_{j+2}, which again makes the equations in the blocks Y_{j-2} and Y_{j+2} linear. Continuing like this we have rapidly solved the whole system.

Similarly, assume the Gröbner basis has solved the Dobbertin blocks at some step. This gives us the solution to all the variables in X_1 which makes the first quasigroup block $Y_1 = X_1 *_{vv}^{i_1} X_2$ linear. Solving this gives us the first half of the equations of the block Y_2 and so on. This means that the solution of the whole MQQ system is reduced to either solving just one block of quasigroup equations, or solving the Dobbertin transformation. The security of solving the MQQ system is therefore the minimum complexity of solving Dobbertin transformation and one MQQ block.

5 Further analysis of MQQ

Using the knowledge from Section 4.2 we investigate if it is possible to strengthen the MQQ system. To do this we have to determine the weakest part of the system; the Dobbertin transformation or the quasigroup transformation.

5.1 The Dobbertin transformation

Recall that the Dobbertin transformation is a bijection over $\mathbb{Z}_2^{2r+1}$ defined by the function $D_r(x) = x^{2^{r+1}+1} + x^3 + x$. For any r we can view this function as $2r + 1$ Boolean functions in $2r + 1$ variables. In Table 2 our experimental results on the degree of regularity for solving this system of equations is listed for various r. From the table we see that the degree of regularity for the Dobbertin transformation seems to be bounded from above by 3. This means Dobbertin's transformation is not "random" and can be easily solved by Gröbner bases computation. In addition we learn that tweaking the MQQ system by increasing the size of the extension field, over which the transformation is defined, will have no effect on strengthening the system.

Proving mathematically (if true) that the degree of regularity for $D_r(x)$ is constant at 3 for all r is difficult. We can however give show that the degree of regularity is low for all practical r. Let $\mathbb{K} = \mathbb{F}_q$ be a field of q elements, and let $\mathbb{L}$ be an extension of degree n over $\mathbb{K}$. Recall that an HFE polynomial f is a low-degree polynomial over $\mathbb{L}$ with the following shape:

$$f(x) = \sum_{\substack{0 \le i,j \le n \\ q^i + q^j \le d}} a_{i,j} x^{q^i+q^j} + \sum_{\substack{0 \le k \le n \\ q^k \le d}} b_k x^{q^k} + c, \qquad (10)$$

Table 2. The observed degree of regularity, D_{reg}, for the Dobbertin bijection over $\mathbb{Z}_2^{2r+1}$ for $r = 2, \ldots, 22$ when computed with magma 2.16-1's implementation of the F_4 algorithm on a 4 processor Intel Xeon 2.93GHz computer with 128GB of memory

r	D_{reg}	r	D_{reg}	r	D_{reg}
2	3	9	3	16	3
3	3	10	3	17	3
4	3	11	3	18	3
5	3	12	3	19	3
6	3	13	3	20	3
7	3	14	3	21	3
8	3	15	3	22	3

where $a_{i,j}, b_k$ and c all lie in $\mathbb{L}$. The maximum degree d of the polynomial has to be chosen such that factorization over $\mathbb{L}$ is efficient [8]. Setting $q = 2$ and $n = 2r + 1$ we notice that the Dobbertin transformation is actually an HFE polynomial, $D_r(x) = x^{2^{r+1}+2^0} + x^{2^1+2^0} + x^{2^0}$. This is very helpfull since a lot of work has been done on the degree of regularity for Gröbner basis compuation of HFE polynomials [16,8]. Faugère and Joux showed that the degree of regularity for Gröbner bases computation of an HFE polynomial of degree d is bounded from above by $\log_2(d)$ [16,17]. For the Dobbertin transformation this means the degree of regularity is bounded from above by $r + 1$.

However, since the coefficients of the Dobbertin transformation all lie in $GF(2)$, we can give an even tighter bound on the degree of regularity. Similar to the weak-key polynomials in [8] the Dobbertin transformation commutes with the Frobenius automorphism and its iterates $F_i(x) : x \mapsto x^{2^i}$ for $0 \leq i \leq n$.

$$D_r \circ F_i(x) = F_i \circ D_r(x) \tag{11}$$

From the equations we see that when $D_r(x) = 0$ we have $F_i \circ D_r(x) = 0$. This means for each i we can add the n equations over $GF(2)$ corresponding to the equation $D_r \circ F_i(x) = 0$ over $GF(2^n)$ to the ideal. However, many of these equations are similar. Actually, we have that F_i and F_j are similar if and only if $gcd(i, n) = gcd(j, n)$ [8]. Worst case scenario is when n is prime. The Frobenius automorphism then gives us $2n$ equations in n variables. From [3] we have the following formula for the degree of regularity for a random system of multivariate equations over $GF(2)$ when the number of equations m is a multiple of the number of variables n. For $m = n(N + o(1))$ with $N > 1/4$ the degree of regularity is

$$\frac{D_{\text{reg}}}{n} = \frac{1}{2} - N + \frac{1}{2}\sqrt{2N^2 - 10N - 1 + 2(n+2)\sqrt{N(N+2)}} + o(1) \tag{12}$$

Setting $N = 2$ we get $D_{\text{reg}} = -\frac{3}{2} + \frac{1}{2}\sqrt{-13 + 16\sqrt{2}} \cdot n \approx 0.051404 \cdot n$. This is the upper bound for a *random* multivariate system with the same number of equations and variables as the Dobbertin transformation. This provides us a good indication that the degree of regularity for Dobbertin (which is not random at all) should be small, as observed in the experiments, and even smaller than a regular HFE polynomial.

5.2 The quasigroup transformation

To get an idea how strong the quasigroup transformation is, we decided to run some experiments where we replaced the input to the Dobbertin transformation with random linear equations. This means that solving the Dobbertin block will no longer make all the equations in the quasigroup transformation linear. The result of our experiment on this special MQQ system where the linear equations are perfectly masked is listed in Table 3. From the table it appears that both the quasigroup size and the number of variables have an effect on the degree of regularity. This tells us that if we replace Dobbertin transformation with a stronger function, the MQQ system can possibly be made strong enough to resist pure Gröbner attack for correct choices of quasigroups size and number of variables.

Table 3. Effects of quasigroup size and the Dobbertin transformation on the observed degree of regularity for Gröbner bases computations of 60 variable MQQ systems. D_{reg} is the observed degree of regularity of normal MQQ systems, while D^*_{reg} is the observed degree of regularity for the same system where the input to Dobbertin has been replaced with random linear equations.

Variables	Quasigroup size	Quasigroups type	Dobbertin	D_{reg}	D^*_{reg}
30	2^5	4 $\text{Quad}_3^s\text{Lin}_2^s$ and 1 $\text{Quad}_2^s\text{Lin}_3^s$	7,9	3	3
	2^{10}	2 $\text{Quad}_8^s\text{Lin}_2^s$	7,7	3	4
40	2^5	5 $\text{Quad}_3^s\text{Lin}_2^s$ and 2 $\text{Quad}_2^s\text{Lin}_3^s$	7,7,7	3	4
	2^{10}	3 $\text{Quad}_8^s\text{Lin}_2^s$	7,9	3	4
	2^{20}	1 $\text{Quad}_{17}^s\text{Lin}_3^s$	7,7,9	3	4
50	2^5	9 $\text{Quad}_3^s\text{Lin}_2^s$	7,7,9	3	3
	2^{10}	4 $\text{Quad}_8^s\text{Lin}_2^s$	9,9	3	4
60	2^5	11 $\text{Quad}_3^s\text{Lin}_2^s$	9,9,9	3	3
	2^{10}	4 $\text{Quad}_8^s\text{Lin}_2^s$ and 1 $\text{Quad}_7^s\text{Lin}_3^s$	7,7,7	3	5
	2^{20}	1 $\text{Quad}_{18}^s\text{Lin}_2^s$ and 1 $\text{Quad}_{17}^s\text{Lin}_3^s$	7,9,9	3	5

6 Conclusion

We have confirmed the results of [27] showing that the degree of regularity for MQQ systems are bounded from above by a small constant, and therefore MQQ systems in large number of variables can easily be broken with Gröbner bases cryptanalysis. The main result of this paper is our explanation of the underlying reason for this bound on the degree of regularity. We explained this by showing how the complexity of solving MQQ systems with Gröbner bases is equal to the minimum of the complexity of solving the Dobbertin transformation and the complexity of solving one MQQ block. Furthermore, our experimental data showed that the degree of regularity for solving the Dobbertin transformation is bounded from above by 3, the same as the bound on the MQQ system. It is natural to conclude that the Dobbertin transformation is a serious weakness in the MQQ system.

We also showed that if the Dobbertin transformation is replaced with an ideal function, which perfectly hides the linear parts of the system, the degree of regularity varies in the size of the quasigroups and the number of variables. We conclude that if a suitable replacement for the Dobbertin transformation is found, MQQ can possibly be made strong enough to resist pure Gröbner attack for correct choices of quasigroups size and number of variables.

References

1. Magali Bardet. *Étude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie.* PhD thesis, Université de Paris VI, 2004.
2. Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. Complexity study of Gröbner basis computation. Technical report, INRIA, 2002. `http://www.inria.fr/rrrt/rr-5049.html`.
3. Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. Complexity of Gröbner basis computation for semi-regular overdetermined sequences over F2 with solutions in F2. Technical report, Institut national de recherche en informatique et en automatique, 2003.
4. Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations. In *Proc. International Conference on Polynomial System Solving (ICPSS)*, pages 71–75, 2004.
5. Magali Bardet, Jean-Charles Faugère, Bruno Salvy, and Bo-Yin Yang. Asymptotic behaviour of the degree of regularity of semi-regular polynomial systems. In *Proc. of MEGA 2005, Eighth International Symposium on Effective Methods in Algebraic Geometry*, 2005.
6. Olivier Billet and Jintai Ding. Overview of cryptanalysis techniques in multivariate public key cryptography. In Massimiliano Sala, Teo Mora, Ludovic Perret, Shojiro Sakata, and Carlo Traverso, editors, *Gröbner bases, coding and cryptography*, pages 263–283. Springer Verlag, 2009.
7. Andrey Bogdanov, Thomas Eisenbarth, Andy Rupp, and Christopher Wolf. Time-Area Optimized Public-Key Engines: MQ -Cryptosystems as Replacement for Elliptic Curves? In *Cryptographic Hardware and Embedded Systems (CHES)*, volume 5154, pages 145–61. Lecture Notes in Computer Science, 2008.
8. Charles Bouillaguet, Pierre-Alain Fouque, Antoine Joux, and Joana Treger. A family of weak keys in hfe (and the corresponding practical key-recovery). Cryptology ePrint Archive, Report 2009/619, 2009.
9. Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal.* PhD thesis, Leopold-Franzens University, 1965.
10. Yanling Chen, Svein Johan Knapskog, and Danilo Gligoroski. Multivariate Quadratic Quasigroups (MQQ): Construction, Bounds and Complexity. Submitted to ISIT 2010, 2010.

11. David Cox, John Little, and Donal O'Shea. *Using Algebraix Geometry.* Springer, 2005.
12. Francoise Levy dit Vehel, Maria Grazia Marinari, Ludovic Perret, and Carlo Traverso. A survey on polly cracker system. In Massimiliano Sala, Teo Mora, Ludovic Perret, Shojiro Sakata, and Carlo Traverso, editors, *Gröbner bases, coding and cryptography*, pages 263–283. Springer Verlag, 2009.
13. Hans Dobbertin. One-to-one highly nonlinear power functions on GF(2^n). *Appl. Algebra Eng. Commun. Comput.*, 9(2):139–152, 1998.
14. Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (F_4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, June 1999.
15. Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, New York, 2002. ACM.
16. Jean-Charles Faugère and Antoine Joux. Algebraic cryptanalysis of Hidden Field Equation (HFE) cryptosystems using Gröbner bases. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *LNCS*, pages 44–60. Springer, 2003.
17. Pierre-Alain Fouque, Gilles Macario-Rat, and Jacques Stern. Key recovery on hidden monomial multivariate schemes. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 2008.
18. Danilo Gligoroski, Smile Markovski, and Svein Johan Knapskog. Multivariate quadratic trapdoor functions based on multivariate quadratic quasigroups. In *MATH'08: Proceedings of the American Conference on Applied Mathematics*, pages 44–49, Stevens Point, Wisconsin, USA, 2008. World Scientific and Engineering Academy and Society (WSEAS).
19. Louis Goubin, Nicolas T. Courtois, and Schlumbergersema Cp. Cryptanalysis of the ttm cryptosystem. In *Advances of Cryptology, Asiacrypt2000*, pages 44–57. Springer, 2000.
20. Louis Granboulan, Antoine Joux, and Jacques Stern. Inverting HFE is quasipolynomial. In *CRYPTO*, pages 345–356, 2006.
21. Aviad Kipnis, Hamarpe St. Har Hotzvim, Jacques Patarin, and Louis Goubin. Unbalanced oil and vinegar signature schemes. In *In Advances in Cryptology EUROCRYPT 1999*, pages 206–222. Springer, 1999.
22. Aviad Kipnis and Adi Shamir. Cryptanalysis of the oil & vinegar signature scheme. In *CRYPTO '98: Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, pages 257–266, London, UK, 1998. Springer-Verlag.
23. MAGMA. High performance software for algebra, nuber theory, and geometry — a large commercial software package. http://magma.maths.usyd.edu.au.
24. Smile Markovski. Quasigroup string processing and applications in cryptography. In *Proc. 1-st Inter. Conf. Mathematics and Informatics for industry MII 2003, 1416 April, Thessaloniki, 278290*, page 278290, 2003.
25. Smile Markovski, Danilo Gligoroski, and Verica Bakeva. Quasigroup string processing. In *Part 1, Contributions, Sec. Math. Tech. Sci., MANU, XX*, pages 1–2, 1999.
26. Tsutomu Matsumoto and Hideki Imai. Public quadratic polynomial-tuples for efficient signature-verification and message-encryption. In *Advances in Cryptology – EUROCRYPT 1988*, volume 330 of *LNCS*, pages 419–453. Springer–Verlag, 1988.
27. Mohamed Saied Mohamed, Jintai Ding, Johannes Buchmann, and Fabian Werner. Algebraic attack on the MQQ public key cryptosystem. In *CANS '09: Proceedings*

of the 8th International Conference on Cryptology and Network Security, pages 392–401, Berlin, Heidelberg, 2009. Springer-Verlag.
28. Jacques Patarin. Cryptanalysis of the matsumoto and imai public key scheme of eurocrypt'88. In *Lecture Notes in Computer Science*, pages 248–261, 1995.
29. Jacques Patarin. Hidden field equations (hfe) and isomorphisms of polynomials (ip): two new families of asymmetric algorithms. In *EUROCRYPT*, pages 33–48. Springer-Verlag, 1996.
30. Jacques Patarin. The oil & vinegar signature scheme, 1997.
31. Jacques Patarin, Louis Goubin, and Nicolas Courtois. C $^* - +$ and hm: Variations around two schemes of t.matsumoto and h.imai. In *Advances in Cryptology - Asiacrypt'98*, volume 1514, pages 35–49. Springer, 1998.
32. Adi Shamir. Efficient signature schemes based on birational permutations. In *CRYPTO '93: Proceedings of the 13th annual international cryptology conference on Advances in cryptology*, pages 1–12, New York, NY, USA, 1994. Springer-Verlag New York, Inc.
33. J. D. H. Smith. *An introduction to quasigroups and their representations.* Chapman & Hall/CRC, 2007.
34. 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.

A Algorithm for generating random MQQ

In this section we present the pseudo-code for how the MQQs used in this paper have been generated. The code was implemented in magma.

Algorithm *MQQ algorithm*
```
1.  $n \leftarrow$ {size of quasigroup}
2.  $L \leftarrow$ {number of linear terms}
3.  if $L \leq 2$
4.     then $Q = n$
5.     else $Q = n - L$
6.  CorrectDeg $\leftarrow$ True
7.  while CorrectDeg
8.     do $A1 \leftarrow$ IdentityMatrix($n$) (* The identity matrix of size $n$ *)
9.        $X1 \leftarrow [x_1, \ldots, x_n]^T$
10.       $X2 \leftarrow [x_{n+1}, \ldots, x_{2n}]^T$
11.       for $i \leftarrow 1$ to $Q$
12.           do for $j \leftarrow i+1$ to $n$
13.                  do for $k \leftarrow i+1$ to $(n)$
14.                         $r \in_R \{0,1\}$ (* random element from the set
                            {0,1} *)
15.                         $A1_{(i,j)} = A1_{(i,j)} + r * X1_k$
16.       $B \leftarrow$ RandomNonSingularBooleanMatrix($n$) (* Random non singular
          Boolean matrix of size $n$ *)
17.       $C \leftarrow$ RandomBooleanVector(n) (* Random Boolean vector of size
          $n$ *)
```

```
18.       $A1$ ←$B * A1$
19.       $X1$ ←$B * X1 + C$
20.       $L1$ ←RandomNonSingularBooleanMatrix($n$) (∗ Random non singu-
          lar Boolean matrix of size $n$ ∗)
21.       $L2$ ←RandomNonSingularBooleanMatrix($n$) (∗ Random non singu-
          lar Boolean matrix of size $n$ ∗)
22.       $A1$ ←LinTrans($A1, L1$) (∗ Lineary transform the indeterminates of
          $A1$ according to $L1$ ∗)
23.       $X1$ ←LinTrans($X1, L1$) (∗ Lineary transform the indeterminates of
          $X1$ according to $L1$ ∗)
24.       $X2$ ←LinTrans($X2, L2$) (∗ Lineary transform the indeterminates of
          $X2$ according to $L2$ ∗)
25.       MQQ ←$A1 * X2 + X1$
26.       GBMQQ ←Gröbner(MQQ,2) (∗ The truncated Gröbnerbasis of de-
          gree 2 under graded reverse lexicographical ordering. ∗)
27.       Deg ←{number of linear terms in GBMQQ}
28.       **if** Deg= $L$
29.         **then** CorrectDeg ←False
30. **return** GBMQQ
```

Multivariate Quasigroups Defined by T-functions

S. Samardziska[1], S. Markovski[1], D. Gligoroski[2]

[1] Institute of Informatics, Faculty of Sciences, "Ss Cyril and Methodius" University, Skopje, Republic of Macedonia, {simona,smile}@ii.edu.mk
[2] Faculty for Informatics, Institute of Telematics, Norwegian University of Science and Technology, Trondheim, Norway, danilog@item.ntnu.no

Abstract. In this paper we investigate quasigroups represented as vector valued Boolean functions in their polynomial ANF form. We propose an effective general deterministic procedure for construction of multivariate quasigroups of arbitrary order and degree. It offers solution to some open problems regarding the production of Multivariate Quadratic Quasigroups (MQQ) used in the MQQ PKC scheme, of orders higher than 2^5 and of desired type, as well as finding a lower bound on the number of MQQs. Even more, it provides deeper insight into the structure of the MQQs and enables a different classification concerning their complexity. For this, we use T-functions, and make a complete characterization of the T-functions that define permutations and quasigroups.

Keywords: Multivariate Quasigroup, Multivariate Quadratic Quasigroup, T-function

1 Introduction

Using quasigroups for cryptographic purposes is an idea present in the scientific public for several decades and is becoming more exploited and popular every day. Here we are interested in quasigroups represented as vector valued Boolean functions (v.v.b.f.) in their polynomial ANF form, and the problem of finding a procedure for constructing quasigroups of any order and degree. Such procedure would be especially suitable for creation of Multivariate Quadratic Quasigroups (MQQ) that are the basis of the public key cryptsystem proposed by Gligoroski at al. [2] in 2007, as an attempt to overcome the weaknesses of the previously proposed MQ schemes. Although the encryption/decryption part of MQQ was broken by Mohamed et al. [4] in 2008, by removing 1/4 of the public key equations the scheme can still be used for digital signatures, keeping its ultra-fast performances for signing and verifications.

In this paper we propose an effective general deterministic procedure for construction of multivariate quasigroups of arbitrary order and degree. It offers solution to the open problems of producing quasigroups of orders higher than 2^5, mentioned in [2]. Also, it provides deeper insight into the structure of the MQQs and enables a different classification concerning the complexity of the MQQs, with great security implications.

Our procedure uses T-functions defined by Klimov and Shamir [3].

We will not distinguish between the elements $x \in \mathbb{Z}_{2^w}$ and their binary representation $(x_w, x_{w-1}, \ldots, x_1) \in \mathbb{Z}_2^w$. If $f : (\mathbb{Z}_2^w)^m \to (\mathbb{Z}_2^w)^l$ and x is a m-coordinate vector of w-bit words, let x_i^j denote the i-th bit of the j-th component of x, and let $f(x)_i^j$ denote the i-th bit of the component j of $f(x)$.

Definition 1. *Let $f : (\mathbb{Z}_2^w)^m \to (\mathbb{Z}_2^w)^l$. f is called T-function, if for every $x \in (\mathbb{Z}_2^w)^m$, the k-th bit of the j-th component, $f(x)_k^j$, depends only on the rightmost k bits of each component of x, for every $j \in \{1, \ldots, m\}$.*

Proposition 1. [3] *The Boolean functions: complement, "and", "exclusive or", and "or", and the operations: negation, addition, subtraction and multiplication over $\mathbb{Z}_2^w$ are all T-functions.*

Note that left shift is a T-function (since it is equivalent to multiplication by a power of 2), but right shift and circular rotations are not. Also, composition of two T-functions is again a T-function, and thus every sequence of T-functions applied to $x \in (\mathbb{Z}_2^w)^m$ is also a T-function.

2 T-functions that define permutations and quasigroups

Klimov and Shamir [3] proposed sufficient, but not necessary, conditions for a T-function to be a permutation or a quasigroup.

Here we provide sufficient and necessary conditions, and make a complete characterization of the T-functions that define permutations and quasigroups.

Let $w \geq r \geq 1$, and let $f : \mathbb{Z}_2^w \to \mathbb{Z}_2^r$ be a vector valued Boolean function. f can be represented as an r-tuple of Boolean functions $f = (f^{(r)}, f^{(r-1)}, \ldots, f^{(1)})$, where $f^{(s)} : \mathbb{Z}_2^w \to \mathbb{Z}_2$, $s = 1, \ldots, r$, and $f^{(s)}(x)$ is the s-th bit of $f(x)$. The Boolean function $f^{(s)}(x_w, \ldots, x_1)$ can be represented by its Algebraic Normal Form (ANF) as a polynomial in w variables $x_1, \ldots, x_w$ of the form

$$f^{(s)}(x_w, \ldots, x_1) = \bigoplus_{j=(j_w,\ldots,j_1)\in\mathbb{Z}_2^w} a_j x_w^{j_w} \ldots x_2^{j_2} x_1^{j_1},$$

where $a_j \in \mathbb{Z}_2$, x^0 is an empty string and $x^1 = x$. The algebraic degree of a Boolean function f is the number of variables in the longest term of the ANF form of f. Generally, higher degree means more applicable Boolean function for cryptography.

It can be verified that the ANF form of a T-function $f : \mathbb{Z}_2^w \to \mathbb{Z}_2^w$, is $f = (f^{(w)}, f^{(w-1)}, \ldots, f^{(1)})$, where, for every $s = 1, \ldots, w$,

$$f^{(s)}(x_w, \ldots, x_1) = \bigoplus_{j=(j_s,\ldots,j_1)\in\mathbb{Z}_2^s} a_j x_s^{j_s} \ldots x_1^{j_1}.$$

Lemma 1. *Let $p = (p^{(w)}, p^{(w-1)}, \ldots, p^{(1)})$ be a T-function that is a permutation on $\mathbb{Z}_2^w$. Then, for every $m = 1, \ldots, w$, the projection $p_{|m} = (p^{(m)}, p^{(m-1)}, \ldots, p^{(1)})$ is a permutation on $\mathbb{Z}_2^m$.*

Proof. We first prove that $p_{|w-1}$ must be a permutation too. Suppose that this is not true. Then there are different $(x_{w-1},\dots,x_1)$ and $(y_{w-1},\dots,y_1)$ in $\mathbb{Z}_2^{w-1}$ such that $p_{|w-1}(x_{w-1},\dots,x_1) = p_{|w-1}(y_{w-1},\dots,y_1)$. Since p is a T-function, $p_{|w-1}(a_{w-1},\dots,a_1) = (p^{(w-1)}(a_w,\dots,a_1),\dots,p^{(1)}(a_w,\dots,a_1))$ for any $a \in \mathbb{Z}_2^w$. Now, for $(0,x_{w-1},\dots,x_1)$, $(1,x_{w-1},\dots,x_1)$, $(0,y_{w-1},\dots,y_1)$, $(1,y_{w-1},\dots,y_1)$ in $\mathbb{Z}_2^w$ which are all different, we have that the last $w-1$ bits of $p(0,x_{w-1},\dots,x_1)$, $p(1,x_{w-1},\dots,x_1)$, $p(0,y_{w-1},\dots,y_1)$ and $p(1,y_{w-1},\dots,y_1)$ are the same. But there are only two possible values for $p^{(w)}$, 0 or 1, which implies that p maps these four different numbers into only two. This contradicts the fact that p is a permutation. By induction, $p_{|m} = (p^{(m)},p^{(m-1)},\dots,p^{(1)})$ is a permutation on $\mathbb{Z}_2^m$ for every $m = 1,\dots,w$.

A finite Boolean function, besides the representation in ANF form, can be represented by its truth table. It is called balanced, if there is an equal number of ones and zeros in its truth table.

Lemma 2. [1]*Let $f = (f^{(w)},f^{(w-1)},\dots,f^{(1)})$ be a Boolean function from $\mathbb{Z}_2^w$ to $\mathbb{Z}_2^w$. f is a permutation if and only if every nonzero linear combination $a_w f^{(w)} \oplus a_{w-1}f^{(w-1)} \oplus \cdots \oplus a_1 f^{(1)}$ is a balanced Boolean function.*

The following Theorem is one of the main results in this paper.

Theorem 1. *A Boolean T-function $p = (p^{(w)},p^{(w-1)},\dots,p^{(1)})$ from $\mathbb{Z}_2^w$ to $\mathbb{Z}_2^w$ is a permutation if and only if for every $s = 1,\dots,w$, the component $p^{(s)}$ is of the form*

$$p^{(s)}(x_w,\dots,x_1) = x_s \oplus \left(\bigoplus_{j=(j_{s-1},\dots,j_1)\in\mathbb{Z}_2^{s-1}} a_j x_{s-1}^{j_{s-1}} \dots x_2^{j_2} x_1^{j_1} \right), \qquad (1)$$

Proof. Let p be a permutation Boolean T-function over $\mathbb{Z}_2^w$. Since p is a T-function, for every $s = 1,\dots,w$,

$$p^{(s)}(x_w,\dots,x_1) =$$
$$= x_s \left(\bigoplus_{j=(j_{s-1},\dots,j_1)\in\mathbb{Z}_2^{s-1}} b_j x_{s-1}^{j_{s-1}} \dots x_1^{j_1} \right) \oplus \left(\bigoplus_{j=(j_{s-1},\dots,j_1)\in\mathbb{Z}_2^{s-1}} a_j x_{s-1}^{j_{s-1}} \dots x_1^{j_1} \right) =$$
$$= x_s \cdot B_s \oplus \left(\bigoplus_{j=(j_{s-1},\dots,j_1)\in\mathbb{Z}_2^{s-1}} a_j x_{s-1}^{j_{s-1}} \dots x_1^{j_1} \right).$$

We show that $B_s \equiv 1$ for every $s = 1,\dots,w$.

Suppose there is some $s_1 \in \{1,\dots,w\}$ such that $B_{s_1} \not\equiv 1$. This means that there is a (s_1-1)-tuple of bits $(\alpha_{s_1-1},\dots,\alpha_1)$ such that $B_{s_1} = 0$. But then $p_{|s_1}(0,\alpha_{s_1-1},\dots,\alpha_1) = p_{|s_1}(1,\alpha_{s_1-1},\dots,\alpha_0)$, i.e., $p_{|s_1}$ is not a permutation, which contradicts Lemma 1. Therefore, $B_s \equiv 1$ for every $s = 1,\dots,w$.

Conversely, let p be a Boolean function of the form (1).

Let $a_w p^{(w)} \oplus a_{w-1} p^{(w-1)} \oplus \cdots \oplus a_1 p^{(1)}$ be an arbitrary nonzero linear combination of the coordinates of p, and let m be the highest index such that $a_m = 1$. Then

$$a_w p^{(w)} \oplus a_{w-1} p^{(w-1)} \oplus \cdots \oplus a_1 p^{(1)} = x_m \oplus \left(\bigoplus_{j=(j_{m-1},\ldots,j_1)\in \mathbb{Z}_2^{m-1}} \beta_j x_{m-1}^{j_{m-1}} \ldots x_1^{j_1} \right).$$

For each variation of the bits $x_w, \ldots, x_{m+1}, x_{m-1}, \ldots, x_1$, the bit x_m can be 0 or 1, so the last sum, for exactly half of the elements of $\mathbb{Z}_2^w$ is 0 and for the other half it is 1. Therefore, $a_w p^{(w)} \oplus a_{w-1} p^{(w-1)} \oplus \cdots \oplus a_1 p^{(1)}$ is balanced, so from Lemma 2, we get that p is a permutation.

Now that we have characterized the permutation T-functions, the properties of the Boolean functions that define quasigroups, are quite clear.

Theorem 2. *A Boolean T-function $q : (\mathbb{Z}_2^w)^2 \to \mathbb{Z}_2^w$, defines a quasigroup if and only if it is of the form $q = (q^{(w)}, q^{(w-1)}, \ldots, q^{(1)})$ where for every $s = 1, \ldots, w$, and $(x, y) = (x_w, \ldots, x_1;\ y_w, \ldots, y_1)$,*

$$q^{(s)}(x, y) = x_s \oplus y_s \oplus \left(\bigoplus_{\substack{j = (j_{s-1},..,j_1) \in \mathbb{Z}_2^{s-1} \\ k = (k_{s-1},..,k_1) \in \mathbb{Z}_2^{s-1}}} b_{jk} x_{s-1}^{j_{s-1}} \ldots x_1^{j_1} y_{s-1}^{k_{s-1}} \ldots y_1^{k_1} \right). \quad (2)$$

Proof. Let q be a function in the given form. It is enough to show that for a given $a = (a_{w-1}, \ldots, a_0) \in \mathbb{Z}_2^w$, $q(x, a)$ and $q(a, y)$ are permutations.

$$q(x, a) = (q^{(w-1)}(x, a), q^{(w-2)}(x, a), \ldots, q^{(0)}(x, a))$$

and for every $s = 0, \ldots, w-1$,

$$q^{(s)}(x, a) = x_s \oplus a_s \oplus \left(\bigoplus_{\substack{j = (j_{s-1},..,j_0) \in \mathbb{Z}_2^{s} \\ k = (k_{s-1},..,k_0) \in \mathbb{Z}_2^{s}}} b_{jk} x_{s-1}^{j_{s-1}} \ldots x_1^{j_1} x_0^{j_0} a_{s-1}^{k_{s-1}} \ldots a_1^{k_1} a_0^{k_0} \right).$$

From Theorem 1 the last is a permutation. Similarly, we prove that $q(a, y)$ is a permutation as well.

Conversely, let q define a quasigroup. Then, for every $a \in \mathbb{Z}_2^w$, $q(x, a)$ and $q(a, y)$ are permutations. Again, from Theorem 1 the coefficient of x_s in $q^{(s)}(x, a)$ is identically equal to 1, and the bit x_s does not affect the rest of the sum. The same holds for the coefficient of y_s in $q^{(s)}(a, y)$, i.e. it is identically equal to 1, and y_s does not affect the rest of the sum. This is only possible if $q^{(s)}(x, y)$ is of the form (2).

All quasigroups that are T-functions are very structured. In general, they have the required classical properties for application in cryptography, like noncommutativity, nonassociativity, nonidempotency, nonlinearity and so on. Still, cryptoprimitives using them can be attacked quite easy, for example, using Hensel lifting.

Nevertheless, because of their simple shape, huge number and clear properties, they can be used as a base for fast creation of quasigroups with solid cryptographic properties.

In [7], Wu proves that affine transformations of the components or of the variable of a Boolean permutation, produce new Boolean permutations. They can be used to efficiently mix the bits of a Boolean permutation.

We say that a quasigroup is a Boolean quasigroup if it is defined as vector valued Boolean function. The results of Wu can be applied on Boolean quasigroups, too. The proof that it is possible, basically reduces to considering one of the quasigroup variables as a parameter. Then, clearly, we have a permutation in the other variable. From the definition of quasigroups, since the new functions are permutations, we have the sufficient condition for obtaining a quasigroup.

Proposition 2. *Let $q = (q^{(w)}, q^{(w-1)}, \dots, q^{(1)})$ be a Boolean quasigroup over $\mathbb{Z}_2^w$, let $\mathbf{D} = (d_{ij})$ be a $w \times w$ binary matrix and let $c = (c_w, \dots, c_1) \in \mathbb{Z}_2^w$. Then $q\mathbf{D} \oplus c$ is a Boolean quasigroup if and only if $\mathbf{D}$ is nonsingular.*

Proposition 3. *Let $q = (q^{(w)}, q^{(w-1)}, \dots, q^{(1)})$ be a Boolean quasigroup over $\mathbb{Z}_2^w$, let $\mathbf{D_1} = (d_{ij}^1)$ and $\mathbf{D_2} = (d_{ij}^2)$ be $w \times w$ binary matrices and let $c_1 = (c_w^1, \dots, c_1^1)$, $c_2 = (c_w^2, \dots, c_1^2) \in \mathbb{Z}_2^w$. Then $q(x\mathbf{D_1} \oplus c_1, y\mathbf{D_2} \oplus c_2)$ is a Boolean quasigroup if and only if $\mathbf{D_1}$ and $\mathbf{D_2}$ are nonsingular.*

Note that the transformations in the last two propositions are actually isotopies.

3 Construction of Boolean quasigroups of various degrees and orders

The established form (2) of the T-functions that are quasigroups from the previous section, is quite easy for manipulation. But, for creation of special types of quasigroups we will rewrite it in an equivalent matrix form, very suitable for implementation.

Theorem 3. *Let $x = (x_w, \dots, x_1)$ and $y = (y_w, \dots, y_1)$ be variables over $\mathbb{Z}_2^w$. Then every T-function that is a quasigroup can be written as a vector valued Boolean function in a unique form*

$$q(x_w, \dots, x_1, y_w, \dots, y_1) = \mathbf{A_1} \cdot (x_w, \dots, x_1)^T + \mathbf{A_2} \cdot (y_w, \dots, y_1)^T + b^T, \quad (3)$$

where $\mathbf{A_1} = [f_{ij}]_{w \times w}$ and $\mathbf{A_2} = [g_{ij}]_{w \times w}$ are upper triangular matrices of Boolean expressions, such that:

– *for every $i = 1, \ldots, w$, $f_{ii} = 1$ and $g_{ii} = 1$,*
– *for every $i = 1, \ldots, w-1$, $f_{iw} = f_{iw}(y_1)$, and g_{iw} are constants,*
– *for all i, j, $i < j < w$, f_{ij} can depend only on the variables $x_{w-j}, \ldots, x_1$, $y_{w-j+1}, \ldots, y_1$ and g_{ij} can depend only on $x_{w-j}, \ldots, x_1, y_{w-j}, \ldots, y_1$,*
– *the vector $b = (b_w, \ldots, b_1)$ is a Boolean constant vector.*

Proof. We show that (3) and (2) are equivalent forms of a T-function that is a quasigroup. From (3), q is of the form $q = (q^{(w)}, q^{(w-1)}, \ldots, q^{(1)})$ where for every $s = 1, \ldots, w$, and $(x, y) = (x_w, \ldots, x_1;\ y_w, \ldots, y_1)$,

$$\begin{aligned}
q^{(s)}(x,y) &= (0, \ldots, 0, 1, f_{w-s+1,w-s+2}, \ldots, f_{w-s+1,w}) \cdot (x_w, \ldots, x_1)^T + \\
&+ (0, \ldots, 0, 1, g_{w-s+1,w-s+2}, \ldots, g_{w-s+1,w}) \cdot (y_w, \ldots, y_1)^T + b_s = \\
&= x_s \oplus f_{w-s+1,w-s+2}(x_{s-2}, \ldots, x_1, y_{s-1}, \ldots, y_1) x_{s-1} \oplus \\
&\oplus f_{w-s+1,w-s+3}(x_{s-3}, \ldots, x_1, y_{s-2}, \ldots, y_1) x_{s-2} \oplus \\
&\quad \vdots \\
&\oplus f_{w-s+1,w-1}(x_1, y_2, y_1) x_2 \oplus f_{w-s+1,w}(y_1) x_1 \oplus \\
&\oplus y_s \oplus g_{w-s+1,w-s+2}(x_{s-2}, \ldots, x_1, y_{s-2}, \ldots, y_1) y_{s-1} \oplus \\
&\oplus g_{w-s+1,w-s+3}(x_{s-3}, \ldots, x_1, y_{s-3}, \ldots, y_1) y_{s-2} \oplus \\
&\quad \vdots \\
&\oplus g_{w-s+1,w-1}(x_1, y_1) y_2 \oplus g_{w-s+1,w} y_1 \oplus b_s = \\
&= x_s \oplus x_{s-1} \cdot \left(\bigoplus_{\substack{j = (j_{s-2}, .., j_1) \in \mathbb{Z}_2^{s-2} \\ k = (k_{s-1}, .., k_1) \in \mathbb{Z}_2^{s-1}}} \beta_{jk}^{(s-1)} x_{s-2}^{j_{s-2}} \ldots x_1^{j_1} y_{s-1}^{k_{s-1}} \ldots y_1^{k_1} \right) \oplus \\
&\quad \vdots \\
&\oplus x_2 \cdot \left(\bigoplus_{\substack{j_1 \in \mathbb{Z}_2 \\ k = (k_2, k_1) \in \mathbb{Z}_2^2}} \beta_{jk}^{(2)} x_1^{j_1} y_2^{k_2} y_1^{k_1} \right) \oplus x_1 \cdot \left(\bigoplus_{k_1 \in \mathbb{Z}_2} \beta_k^{(1)} y_1^{k_1} \right) \oplus \\
&\oplus y_s \oplus y_{s-1} \cdot \left(\bigoplus_{\substack{j = (j_{s-2}, .., j_1) \in \mathbb{Z}_2^{s-2} \\ k = (k_{s-2}, .., k_1) \in \mathbb{Z}_2^{s-2}}} \gamma_{jk}^{(s-1)} x_{s-2}^{j_{s-2}} \ldots x_1^{j_1} y_{s-2}^{k_{s-2}} \ldots y_1^{k_1} \right) \oplus \\
&\quad \vdots \\
&\oplus y_2 \cdot \left(\bigoplus_{j_1 \in \mathbb{Z}_2, k_1 \in \mathbb{Z}_2} \gamma_{jk}^{(2)} x_1^{j_1} y_1^{k_1} \right) \oplus y_1 \cdot \gamma^{(1)} \oplus b_s,
\end{aligned}$$

which is equivalent to (2).

Using (3) (or (2)) one can create a quasigroup that is a T-function of arbitrary degree, by restricting the degrees of f_{ij} and g_{ij}. It also allows construction of a quasigroup of arbitrary order 2^w. This characteristic, together with the various types of isotopic transformations given, is especially suitable for creation of MQQs.

As the authors of [2] noted in the paper, the randomized algorithm for generating MQQs given there was able to produce only quasigroups of low orders (at most 2^5). That is why the authors proposed the creation of MQQs of higher order as an open problem. Another important issue mentioned, was distinguishing the different types of MQQs, and finding a way of producing only quasigroups of the desired type. Finally, finding their number, or lower bound was also posed as open research question regarding the security of the algorithm.

Here, we propose an effective general algorithm, that gives answers to these questions, but also goes deeper into the structure of the MQQs and enables a different classification, with great security implications.

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

The general form of a MQQ of order 2^w that is a T-function, follows directly from the general case.

Corollary 1. *The vector valued Boolean function $q(x_w, \dots, x_1, y_w, \dots, y_1)$ over $\mathbb{Z}_2^w$ defines a quasigroup that is a multivariate quadratic quasigroup and a T-function if and only if it can be written in the form (3) where f_{ij} and g_{ij} are linear Boolean expressions.*

We will call these quasigroups Triangular Multivariate Quadratic Quasigroups (T-MQQ).

Proposition 4. *With suitably chosen matrices $\mathbf{A_1}$ and $\mathbf{A_2}$ of linear Boolean expressions, using the form (3), one can produce a T-MQQ of arbitrary type $Quad_{w-k}Lin_k$, for $k = 1, \dots, w$, but never of type $Quad_wLin_0$.*

Proof. In (3), $q^{(s)}$ is of the form

$$\begin{aligned} q^{(s)}(x,y) = (0, \dots, 0, 1, f_{w-s+1,w-s+2}, \dots, f_{w-s+1,w}) \cdot (x_w, \dots, x_1)^T + \\ + (0, \dots, 0, 1, g_{w-s+1,w-s+2}, \dots, g_{w-s+1,w}) \cdot (y_w, \dots, y_1)^T + b_s. \end{aligned}$$

So, if we choose $f_{w-s+1,j}$ and $g_{w-s+1,j}$, $j = w-s+1, \dots, w$, to be constants (0s and 1s), than $q^{(s)}$ will be a linear polynomial. In this manner we can produce as many linear coordinates of q as we want. Note that the structure of a T-function implies that $q^{(1)}$ is always linear, hence a T-MQQ of type $Quad_wLin_0$ can not be constructed.

The type of linear Boolean expressions that are present as elements in the matrices $\mathbf{A_1}$ and $\mathbf{A_2}$, in (3), determines the complexity of the produced T-MQQ

$q = (q^{(w)}, q^{(w-1)}, \dots, q^{(1)})$ in the sense of the different quadratic monomials that will occur in the functions $q^{(s)}$, $1 \le s \le w$. Using similar strategy as in the previous proposition, we have the following one.

Proposition 5. *With suitably chosen matrices* $\mathbf{A_1}$ *and* $\mathbf{A_2}$ *of linear Boolean expressions, using the form* (3), *one can produce T-MQQ* $q = (q^{(w)}, q^{(w-1)}, \dots, q^{(1)})$ *with the following structure of* $q^{(s)}$*:*

- *the only quadratic monomials that appear are of type* $x_i x_j$ $(y_i y_j)$,
- $q^{(s)}$ *contains quadratic monomials only of type* $x_i x_j$ *and* $y_i y_j$,
- $q^{(s)}$ *contains quadratic monomials only of type* $x_i x_j$ *and* $x_i y_j$ $(y_i y_j$ *and* $x_i y_j)$,
- $q^{(s)}$ *contains quadratic monomials of type* $x_i x_j$, $x_i y_j$ *and* $y_i y_j$ *(i.e., of all types).*

The implication of this proposition is that there is a way to mix all the variables $x_w, \dots, x_1, y_w, \dots, y_1$ and increase the complexity of the MQQ. We should note that some of the MQ-based cryptsystems that have been broken ([2], [5]) use multivariate quadratic polynomials that don't mix all the variables, which can impair the security of the system.

Nevertheless, sometimes one needs to trade-off between security and efficiency. Quasigroup based cryptsystems use bijective transformations that include the quasigroup operation in one direction (encryption), and some parastrophic operation in the opposite direction (decryption). Generally, finding the parastrophic operation is a time consuming procedure, especially for quasigroups of higher order. The next proposition gives a special form of a MQQ of arbitrary order that does not mix all the variables, but whose left parastrophe can be easily found. (The case for the right parastrophe is analogous.)

Proposition 6. *Let* $\mathbf{A_1} = [f_{ij}]_{w\times w}$ *and* $\mathbf{A_2} = [g_{ij}]_{w\times w}$ *be upper triangular matrices of linear Boolean expressions, such that:*

- *for every* $i = 1, \dots, w$, $f_{ii} = 1$ *and* $g_{ii} = 1$,
- *for every* $i = 1, \dots, w-1$, f_{iw} *and* g_{iw} *are constants,*
- *for all* $i < j < w$, f_{ij} *and* g_{ij} *can depend only on* $x_{w-j}, \dots, x_1$,
- *the vector* $b = (b_w, \dots, b_1)$ *is a Boolean constant vector.*

Then,

$$q(x_w, \dots, x_1, y_w, \dots, y_1) = \mathbf{A_1} \cdot (x_w, \dots, x_1)^T + \mathbf{A_2} \cdot (y_w, \dots, y_1)^T + b^T,$$

is a quasigroup with left parastrophe $q_\backslash$

$$q_\backslash(x_w, \dots, x_1, y_w, \dots, y_1) = \mathbf{A_2^{-1}} \cdot ((y_w, \dots, y_1)^T - \mathbf{A_1} \cdot (x_w, \dots, x_1)^T - b^T).$$

Proof. Clearly, q is a T-MQQ. Since $\mathbf{A_2}$ is upper triangular $\mathbf{A_2^{-1}}$ exists, and since $\mathbf{A_1}$ and $\mathbf{A_2}$ depend only on the x_j variables, one can verify that

$$q(x_w, \dots, x_1, q_\backslash(x_w, \dots, x_1, y_w, \dots, y_1)) = (y_w, \dots, y_1)^T, \text{ and,}$$
$$q_\backslash(x_w, \dots, x_1, q(x_w, \dots, x_1, y_w, \dots, y_1)) = (y_w, \dots, y_1)^T,$$

i.e., $q_\backslash$ is the left parastrophe of q.

Now, using Proposition 2 and Proposition 3 we can perform isotopic transformations to a T-MQQ and obtain a general MQQ.

Proposition 7. *Let q be a T-MQQ of order 2^w as defined in Corollary 1. Let $\mathbf{D}, \mathbf{D_1}, \mathbf{D_2}$ be $w \times w$ nonsingular Boolean matrices, and let c, c_1, c_2 be Boolean vectors of dimension w. Then*

$$q_*(x_w, \dots, x_1, y_w, \dots, y_1) = q((x_w, \dots, x_1) \cdot \mathbf{D_1} + c_1, (y_w, \dots, y_1) \cdot \mathbf{D_2} + c_2) \cdot \mathbf{D} + c$$

defines a MQQ.

Proposition 7 provides a way for construction of MQQs that are suitable for use in an MQ-based cryptsystems. Their strength for application in a variant of the scheme [2] is being studied by the authors at the moment.

Example 1. We give an example of a T-MQQ of order 2^5 obtained using (3). Then using Proposition 7 we construct a general MQQ of order 2^5.

Let $\mathbf{A_1}$, $\mathbf{A_2}$ be 5×5 matrices given by

$$\mathbf{A_1} = \begin{bmatrix} 1 & x_1 \oplus x_2 \oplus x_3 \oplus y_2 \oplus y_4 & x_2 \oplus y_3 & y_1 & 1 \oplus y_1 \\ 0 & 1 & 1 \oplus x_1 \oplus y_1 \oplus y_2 & x_1 \oplus y_1 & 0 \\ 0 & 0 & 1 & 1 \oplus x_1 \oplus y_2 & y_1 \\ 0 & 0 & 0 & 1 & 1 \oplus y_1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\mathbf{A_2} = \begin{bmatrix} 1 & x_3 \oplus y_2 \oplus y_3 & y_2 & x_1 & 0 \\ 0 & 1 & 1 \oplus x_1 \oplus y_2 & x_1 \oplus y_1 & 1 \\ 0 & 0 & 1 & 1 \oplus y_1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

and let $b = (1, 0, 1, 1, 1)$. Then,

$$q(x_5, \dots, x_1, y_5, \dots, y_1) = \mathbf{A_1} \cdot (x_5, \dots, x_1)^T + \mathbf{A_2} \cdot (y_5, \dots, y_1)^T + b^T =$$

$$= \begin{bmatrix} 1 \oplus x_1 \oplus x_2x_3 \oplus x_1x_4 \oplus x_2x_4 \oplus x_3x_4 \oplus x_5 \oplus x_1y_1 \oplus x_2y_1 \oplus x_1y_2 \oplus \\ \oplus x_4y_2 \oplus x_3y_3 \oplus y_2y_3 \oplus x_3y_4 \oplus x_4y_4 \oplus y_2y_4 \oplus y_3y_4 \oplus y_5 \\ x_1x_2 \oplus x_3 \oplus x_1x_3 \oplus x_4 \oplus y_1 \oplus x_2y_1 \oplus x_3y_1 \oplus x_1y_2 \oplus x_3y_2 \oplus y_1y_2 \oplus \\ \oplus y_3 \oplus x_1y_3 \oplus y_2y_3 \oplus y_4 \\ 1 \oplus x_2 \oplus x_1x_2 \oplus x_3 \oplus y_1 \oplus x_1y_1 \oplus y_2 \oplus x_2y_2 \oplus y_1y_2 \oplus y_3 \\ 1 \oplus x_1 \oplus x_2 \oplus x_1y_1 \oplus y_2 \\ 1 \oplus x_1 \oplus y_1 \end{bmatrix}$$

is a T-MQQ. Now, let

$$\mathbf{D_1} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix}, \mathbf{D_2} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix}, \mathbf{D} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix},$$

and $c_1 = (0,0,1,0,1)$, $c_2 = (0,0,1,1,1)$ and $c = (1,0,0,1,1)$. Then

$$\begin{aligned} &q_*(x_5,\dots,x_1,y_5,\dots,y_1) = \\ &= q((x_5,\dots,x_1)\cdot \mathbf{D_1} + c_1, (y_5,\dots,y_1)\cdot \mathbf{D_2} + c_2)\cdot \mathbf{D} + c = \end{aligned}$$

$$= \begin{bmatrix} \begin{array}{c} x_3 \oplus x_1x_4 \oplus x_2x_4 \oplus x_1y_1 \oplus x_3y_1 \oplus x_5y_1 \oplus x_1y_2 \oplus x_2y_2 \oplus x_5y_2 \oplus \\ \oplus y_1y_2 \oplus y_3 \oplus x_2y_3 \oplus x_3y_3 \oplus x_4y_3 \oplus y_1y_3 \oplus y_4 \oplus x_3y_4 \oplus y_5 \end{array} \\ \begin{array}{c} x_2 \oplus x_4 \oplus x_1x_4 \oplus x_2x_4 \oplus x_5 \oplus x_1y_1 \oplus x_3y_1 \oplus x_5y_1 \oplus x_1y_2 \oplus x_2y_2 \oplus \\ \oplus x_5y_2 \oplus y_1y_2 \oplus x_2y_3 \oplus x_3y_3 \oplus x_4y_3 \oplus y_1y_3 \oplus y_4 \oplus x_3y_4 \oplus y_5 \end{array} \\ \begin{array}{c} 1 \oplus x_2 \oplus x_1x_2 \oplus x_1x_3 \oplus x_4 \oplus x_1x_4 \oplus x_5 \oplus x_1y_1 \oplus x_2y_1 \oplus x_3y_1 \oplus \\ \oplus x_5y_1 \oplus y_2 \oplus x_2y_2 \oplus x_3y_2 \oplus x_5y_2 \oplus x_1y_3 \oplus y_2y_3 \oplus x_1y_4 \end{array} \\ \begin{array}{c} 1 \oplus x_2 \oplus x_1x_2 \oplus x_3 \oplus x_1x_3 \oplus x_4 \oplus x_1x_4 \oplus x_2x_4 \oplus x_1x_5 \oplus x_2y_1 \oplus \\ \oplus x_3y_2 \oplus x_4y_2 \oplus x_5y_2 \oplus y_3 \oplus x_1y_3 \oplus x_2y_3 \oplus x_3y_3 \oplus x_4y_3 \oplus \\ \oplus y_1y_3 \oplus y_2y_3 \oplus y_4 \oplus x_1y_4 \oplus x_3y_4 \oplus y_1y_4 \oplus y_5 \end{array} \\ \begin{array}{c} x_1 \oplus x_2 \oplus x_3 \oplus x_1x_4 \oplus x_5 \oplus x_1x_5 \oplus x_2y_1 \oplus x_3y_1 \oplus x_4y_1 \oplus x_5y_1 \oplus \\ \oplus x_1y_2 \oplus x_4y_2 \oplus x_5y_2 \oplus y_1y_2 \oplus y_4 \oplus y_1y_4 \end{array} \end{bmatrix}$$

is a MQQ of order 2^5.

At the end we give an estimate of the lower bound of different MQQs of order 2^w, that comes as a consequence of the discussion above.

Proposition 8. *There are exactly* $2^{w+\sum_{j=1}^{w-1} j(4w-4j-1)}$ *T-MQQs of order* 2^w.

Proof. For each f_{ij} there are exactly $2^{2(w-j+1)}$ choices, and for each g_{ij} exactly $2^{2(w-j)+1}$ choices. This means that there are $2^{\sum_{j=2}^{w} 2(w-j+1)(j-1)}$ different matrices $\mathbf{A_1}$ and $2^{\sum_{j=2}^{w}[2(w-j)+1](j-1)}$ different matrices $\mathbf{A_2}$. The vector b can be chosen in 2^w ways. Altogether, the number of different T-MQQs of order 2^w is exactly $2^{w+\sum_{j=1}^{w-1} j(4w-4j-1)}$.

The number of T-MQQs for the first few values of w is given in Table 1.

w	2	3	4	5	6	8	9	10	12	14	15	16
T-MQQs	2^5	2^{16}	2^{38}	2^{75}	2^{131}	2^{316}	2^{453}	2^{625}	2^{1090}	2^{1743}	2^{2150}	2^{2616}

Table 1. T-MQQs of order 2^w

This number can be regarded as a lower bound for general MQQs of order 2^w. Even though it is clear that the number of MQQs that can be created using Proposition 7 is much bigger, the claim itself does not specify it. Still, it provides an estimate of the number of different constructions of MQQs. Since there are around $0.28{\cdot}2^{w^2}$ different nonsingular matrices, the total number of constructions is around $0.28^3 \cdot 2^{3w^2+4w+\sum_{j=1}^{w-1} j(4w-4j-1)}$.

References

1. C. Adams and S. Tavares,*The Structured Design of Cryptographically Good S-Boxes*, Journal of Cryptology (1990) 3, pp.27–41
2. D. Gligoroski, S. Markovski, and S.J. Knapskog, *Multivariate quadratic trapdoor functions based on multivariate quadratic quasigroups*, American Conference on Applied Mathematics; Harvard, March 2008, USA.
3. A. Klimov and A. Shamir, *A New Class of Invertible Mappings*, In B.S. Kaliski Jr. and C .K. Koc and C. Paar, editor, 4th Workshop on Cryptographic Hardware and Embedded Systems (CHES), volume , pages 471–484. Springer-Verlag, Lecture Notes in Computer Science, August 2002.
4. M. S. E. Mohamed, J. Ding, J. Buchmann and F. Werner *Algebraic Attack on the MQQ Public Key Cryptsystem*, Lecture Notes in Computer Science, Volume 5888/2009, pp. 392-401
5. A. Kipnis, J. Patarin, and L. Goubin, *Unbalanced Oil and Vinegar signature schemes*, Advances in Cryptology, EUROCRYPT 1999, LNCS Vol. 1592, pp. 206–222, 1999.
6. S. Samardziska, *Polynomial n-ary quasigroups of order p^w*, Masters' thesis, PMF - Skopje, 2009, `http://sites.google.com/site/samardziska/publications/pubs/mastersSamardziska.pdf`
7. C. K. Wu and V. Varadharajan, *Public key cryptsystems based on Boolean permutations and their applications*, International journal of computer mathematics 2000, vol. 74, no2, pp. 167-184

Lattice Polly Cracker Signature (abstract)

Emmanuela Orsini, Carlo Traverso

Dipartimento di Matematica, Università di Pisa.*

1 Introduction

Lattice Polly Cracker (LPC) is a lattice cryptosystem using lattice Gröbner bases. The cryptosystem uses two lattices, algebraically isomorphic but not isometric. The public lattice L has a Gröbner basis that is hard to compute, and the private lattice L' has a Gröbner basis that is simple to compute. The isomorphism constitutes the trapdoor information.

A message m, chosen in a subset $M \subseteq \mathbb{Z}^n$ is encrypted adding to it an element of the public lattice L, and is decrypted computing the normal form of the message modulo the Gröbner basis of L'.

In this paper we give further analysis linking normal form with respect to a suitable Gröbner basis to the approximate Closest Vector Problem. This analysis allows to give a test to estimate the security of an instance of LPC.

We define moreover a signature protocol using LPC; this uses a protocol similar to other lattice signature protocols, the signer being identified by her ability to solve an approximate CVP challenge. LPC instances to use for signatures are different from LPC instances to use for encryption, but their construction follows the same scheme.

This LPC signature scheme, like other lattice signature protocols, is not zero-knowledge, and is subject to the learning attack of Nguyen-Regev[6], but allows a perturbation procedure (different from the NTRU perturbation procedure) that makes this attack impossible.

2 Lattice Gröbner bases and CVP

Let $L \subseteq \mathbb{Z}^n$ be an integer lattice and k a field. Let $X = (x_1, \ldots, x_n)$. Then $k[X]$ is isomorphic to the monoid algebra $k[\mathbb{N}^n]$. Any element $\gamma \in \mathbb{Z}^n$ can be represented as a pair (α, β) of elements of $\mathbb{N}^n$ such that $\alpha - \beta = \gamma$, and the pair (α, β) is unique if they are chosen minimally, i.e. $\alpha_i \beta_i = 0$.

To each $\gamma \in L$ we associate the binomial $X_\gamma = X^\alpha - X^\beta$ (the map is injective unless k has characteristic 2) and the ideal generated by $\{X_\gamma \mid \gamma \in L\}$ is the ideal I_L associated to L (remark that L is uniquely determined by I_L even in characteristic 2).

Given a term-ordering (a total ordering of $\mathbb{Z}^n$ that is compatible with the group law, and such that 0 is the minimum of $\mathbb{N}^n$) the reduced Gröbner basis G_L

* Research performed with the contribute MIUR in the project "Algebra commutativa, computazionale e combinatorica", PRIN 2007

of I_L is defined. Since I_L is a binomial ideal every element of G_L is a binomial $X^\alpha - X^\beta$ and $\alpha_i\beta_i = 0$, hence G_L can be identified with a subset of the lattice and defined purely in lattice terms.

We will restrict ourselves to lattices of full rank (i.e. lattices of rank n in $\mathbb{Z}^n$) although most results are valid more in general (or easy to generalize).

Given a Gröbner basis G of a lattice L, one can compute the normal form $NF(x) = NF_G(x)$ of every element $x \in \mathbb{Z}^n$. We have $(NF(x) = NF(y)) \Leftrightarrow ((x - y) \in L)$. Remark that $NF(x)$ is the smallest such element in $\mathbb{N}^n$, with the comparison done in term-ordering. If the term-ordering is degree-compatible this is also the (possibly non unique) minimal element in l_1 norm, hence the problem solved is NP-hard; this is just an additional proof that finding the Gröbner basis (with respect to a degree-compatible term-ordering) is NP-hard. For other orderings this is not true, for example, for Lex odering finding the Gröbner basis (of a full-rank lattice) is equivalent to finding the Hermite normal form ([2, 3]).

Given a polynomial ideal and its Gröbner basis, the set S of monomials that are in normal form is called the *staircase*. We call *core* of the staircase a rectangle contained in the staircase, and *bounding box* a minimal rectangle containing the staircase (it exists if the ideal is zero-dimensional). More explicitly, a core and the bounding box are defined by two sequences, c_i for the core and b_i for the bounding box, and given $\alpha = (a_1, \ldots, a_n)$ such that $a_i < c_i$ then $\alpha \in S$, and if $\alpha \in S$ then $a_i < b_i$ for all i. The b_i are identified from the reduced Gröbner basis G since G contains a polynomial whose leading term is $x_i^{b_i}$, and the c_i can be checked since the defining condition is equivalent to state that $\prod x_i^{c_i-1}$ is in S.

The core, even if it is maximal, is not unique, but this does not really matter, we are just interested to have staircases containing a core of specified form; the bounding box instead is unique.

In the case of lattices, since the determinant d of a lattice L is equal to the number of elements of $\mathbb{Z}^n/L$, and this in turn is the cardinality of the staircase, we have $\prod c_i \leq d \leq \prod b_i$.

The bounding box limits the size of $NF(x)$ for every $x \in L$, hence a small bounding box means that we can solve the CVP with a good approximation for most vectors.

More on lattice Gröbner bases can be found in [3, 1, 2].

3 Block lattices and their Gröbner bases

We consider a lattice $L \subseteq \mathbb{Z}^n$ of rank n, and a sequence of integers $0 = m_0 < m_1 < \cdots < m_k = n$. Define $n_i = m_i - m_{i-1}$. Define $L_i = L \cap (0^{n-m_i} \oplus \mathbb{Z}^{m_i})$. Each L_i is a sublattice of L of dimension m_i and $L_i \subseteq L_{i+1}$. Define $d_0 = 1$ and $d_i = \det L_i/d_{i-1}$

A basis of L_i can be extended to a basis of L_{i+1}, so eventually we have a basis of L that, repesented as rows of a matrix, and (adding the new vectors at the top) constitutes a block matrix with zero blocks ovder the diagonal; the blocks on the diagonal have determinant $d_k, \ldots, d_1$.

We call such a structure a *block lattice*. Of course, any lattice, given the m_i, has a corresponding block lattice structure, we will restrict our interest to block lattices in with the d_i have some restricted form, for example all the n_i and all the d_i are the same, or vary in some restricted form.

If we choose a block ordering compatible with the block structure (the variables in different blocks compare lexicographically, smaller index being larger), then the Gröbner basis can be computed from the Gröbner basis of the diagonal blocks. Using DegRevLex in the blocks it is easy to obtain “manageable” Gröbner bases (bases with “few” elements and a large core or a small bounding box). Normal form with respect of such bases can be used to provide a good approximation to a closest vector problem; one hence expects that finding such a Gröbner basis is a hard problem.

4 Block Lattice Polly Cracker

Lattice cryposystems have usually a common pattern. There is a set $M \subseteq \mathbb{Z}^n$ and a lattice $L \subseteq \mathbb{Z}^n$ such that M is injectively mapped to $\mathbb{Z}^n/L$. There is some form of trapdoor information that allows to compute the normal form modulo L, i.e. given an element $c = m + l \in M + L$ with $m \in M$ and l in a suitable subset $H \subseteq L$, then m can be recovered. A message is either an element $m \in M$ or an element $h \in H$, and its encryption is $m + h$ (the element that is not the message being chosen at random). Recovering the message is done computing the normal form, and this is only presumed possible through the trapdoor information. For example GGH, NTRU and McEliece fit in this pattern.

Lattice Polly Cracker is a cryptosystem that exploits normal form modulo a lattice Gröbner basis.

While the basic idea is simple, its implementation is harder; recovering a suitable Gröbner basis (moderate cardinality and core sufficiently large) can be done for block lattices, but such lattices are special, and it is easy to discover their structure through the Hermite Normal Form; in particular they have very short vectors (those of the smaller blocks). Hence one defines two lattices, $L \subseteq \mathbb{Z}^n$ and $L' \subseteq \mathbb{Z}^n$ that are algebraically isomorphic: there is an isomorphism $\phi : \mathbb{Z}^n \to \mathbb{Z}^n$ (an unimodular matrix) such that $\phi(L) = L'$. $L \subseteq \mathbb{Z}^n$ is the public lattice, $M \subseteq \mathbb{Z}^n$ is the message set, and L' is a block lattice with a staircase having a core that contains $\phi(M)$. Given a lattice $L \subseteq \mathbb{Z}^n$ whose determinant d factors suitably (e.g. $d = 2^N$) one can exhibit a block lattice $L' \subseteq \mathbb{Z}^n$ algebraically isomorphic to L and with a good staircase (with a suitable core); but it is hard to build one that brings a given set M of messages into the staircase of L'. While it is relatively simple to build L' and the map ϕ, and recovering the lattice L *a posteriori.*

The details of the construction are given in [3], together with heuristic considerations on the difficulty of recovering the shortest block of the private lattice, that has a basis composed of elements of $L : d = \{m \in \mathbb{Z}^n \mid dm \in L\}$, d being the determinant of the smallest block. These elements are conjecturally short vectors of $L : d$.

The security of the public lattice is connected with the length of the shortest vector of the lattice: hence estimating it is important.

We can easily find short vectors in the private lattice: the smallest blocks are composed of short vectors. The map from the private lattice to the public lattice is not of course an isometry, but we expect nonetheless that the image of the smaller blocks contains vectors significantly short.

Given the private key of a LPC instance one can hence compute the shorter vector of the image of some of the smaller blocks, and this allows to estimate how good the instance is, comparing to an estimate of the shorter vector computed through the Gaussian heuristics.This can also be used to estimate the heuristic procedures to define a random LPC instance. It turns out that the procedure outlined in [3] produces public lattices with remarkably large shortest vector. Typically, a random LPC instance of dimension 200 has gaussian estimate 64, and the shortest vector has length from 200 to 500. One of dimension 400 has gaussian estimate 91 and shortest vector length from 900 to 9000.

5 LPC signature (LPCS)

Lattice signatures are usually defined through a path similar to lattice encryption, and is substantially an authentication protocol. The public key is given by a lattice $L \subseteq \mathbb{Z}^n$ and two subsets M and C of $\mathbb{Z}^n$; M represents a commitment to find an element $m \in M$ equivalent modulo L to a challenge $c \in C$. The identification challenge can only be satisfied knowing a trapdoor information. The signature variant implies to respond to a challenge that is a hash of the document to sign.

Signing through LPC requires a setting similar to the LPC encryption, with a public lattice L, a private lattice L' and a trapdoor isomorphism $\phi : \mathbb{Z}^n \to \mathbb{Z}^n$ sending L to L', but now M is a subset containing the inverse image $\phi^{-1}(S)$ of the staircase of L'. To simplify the setting, we build L, ϕ and M such that $M \supseteq \phi^{-1}(B)$, where B is the bounding box.

With procedures similar to those outlined in [3] we show how to build suitable keys. The steps of this construction are the following:

1. Choose the set M of commitments in the form $[0, (r-1)]^n$.
2. Choose the sizes n_i of the diagonal blocks and the determinants d_i; for each i choose randomly a matrix $n_i \times n_i$ having suitably small bounding box. These will be the diagonal blocks of the block matrix defining the private lattice L'. Through these blocks, the staircase S of L' is determined. Let B be the bounding box.
3. Fill randomly the rest of the block diagonal matrix.
4. Choose the isomorphism ϕ^{-1} from $\mathbb{Z}^n$ in the private to the public coordinates, such that $\phi^{-1}(B) \subseteq M$.
5. Check that the public lattice has smallest vector sufficiently large to ensure the desired level of security.

Of course the choice of the parameters has to be made in such a way that the construction is possible and that finding a suitable lattice is reasonably fast. Moreover one can also restrict some choices to get some desirable features (for example, choose the random off-diagonal blocks in a way that ensures with high probability a sufficiently rich group structure of $\mathbb{Z}^n/L$. This can be tested computing the Smith normal form of the lattice).

These steps are the same that are needed to build a LPC instance, but with some significant differences: in LPC we need to consider the core, instead of the bounding box, the inclusion is reversed, and the isomorphism is built in the opposite direction; this has as consequence that it is more difficult to build instances with large shortest vector, although it is still easy to build instances with shortest vector with size larger than the Gaussian heuristics.

6 Suggested parameters and procedures

In this section we make explicit choices of the parameters outlined above. Slightly different choices have been attempted,

With these parameters we are able to build easily instances of LPCS that we believe to be practically secure, although further experience is needed to back this belief.

As for LPC, the key is concealing suitably the smallest block. But while in LPC one has constraints on the map from the public coordinates and the private coordinates (to ensure that the message box is sent inside the core of the staircase) in LPCS one needs to constrain the map from the private coordinates to the public coordinates, to ensure that the bounding box falls inside of the challenge set.

This is a substantial difference, since recovering a basis of the smallest block means recovering small vectors in a private lattice through their image in the public space, that is known, and here we have a constraint on the map, instead of a constraint on the inverse. As a consequence, the public coordinates of the basis of the smallest block of the private lattice are much shorter than than the coordinates of the other basis elements, and this might make retrieving them easier.

Hence to conceal them, ensuring that they are larger than the gaussian heuristic bound for the public lattice requires a richer bag of tricks, and explaining all of them in a short space is rather challenging. We will try to do our best at least showing them,

The first trick is the construction of the private lattice. We choose the blocks, giving special care to the smallest block, that is chosen with special properties.

First, we want that the group $\mathbb{Z}/L$ is as rich as possible, with many subgroups and as far from cyclic as possible, and with a decomposition into a sum of irreducible cyclic groups as non-unique as possible. This is best obtained with a group whose order is a power of 2, hence all the blocks will have determinant a power of 2.

The smallest block will need to be a bit more special than the other blocks, since it is the target of the attacks. and has to be protected. We choose this block of dimension 6, with determinant 1024, bounding box sides between 3 and 6, and bounding box volume at most 4096. The other blocks will have dimension 4, determinant 4096, bounding box sides at most 15 and bounding box volume at most 8192. Finding such blocks by random search is relatively easy.

To complete the private lattice, we have to fill the blocks off the diagonal. We do this randomly, but to ensure that $\mathbb{Z}/L$ is far from cyclic, we put the condition that a suitable number (default 24) of entries to the right of every diagonal block are multiple of a power of 2 (default 8).

As commitment set M we choose the cube $[0, 20]^n$; and we want to find an isomorphism (a unimodular matrix $X = x_{i,j}$) sending the bounding box into M. To simplify, we choose X with non-negative entries. This means that if B is the vector defining the bounding box, BM should be a vector with entries ≤ 20. We pose further bounds that make the construction of a unimodular matrix easier: if $i \leq j$ and $i \leq n - 6$ then $x_{i,i} = 1$ and $x_{i,j} = 0$. Moreover we require that for every $j < n - 6$ at least one $x_{i,j} \neq 0$ with $i \geq n - 6$ (this means that every coodinate mixes with the smallest block, and the last 6 rows have at least one out of 6 elements non zero).

We have checked that these choices ensure that the coordinates of a basis of the smallest block do not have unusually small length, being invariably larger than the predicion of the Gaussian heuristics.

We have not yet experimented enough to conclude that finding the smallest block is hard from the public lattice.

In `http://posso.dm.unipi.it/crypto/LPCS` there are a few challenges. We are able to build LPCS instances lattices of dimension n with determinant 2^{3n} (hence the public key, in Hermite normal form, has $3n^2$ bits) and signatures with $M = (0, 20)^n$ (hence the signature has $4.3n$ bits). We expect LPCS to be at least as secure as other lattice signature schemes with the same lattice dimension.

7 Learning attacks to LPCS and how to avoid them

Like other lattice signature schemes that share the signing mechanism, LPCS is not zero-knowledge. Although slightly different from the signature schemes considered in [6], the parallelogram learning attack appears to be possible on LPCS in its basic form. The parallelogram is no longer complete (the complete parallelogram is the image of the bounding box, and the staircase is only a subset of small density) but identifying the cone generated by the signatures the map ϕ^{-1} may be recovered.

We have not attempted the learning attack, but we have tried a remedy. We have the commitment set M, the staircase S and $\phi^{-1}(S) \subseteq M$. We also have the Gröbner basis G' of L' and we can compute $G = \phi^{-1}(G')$. If $m \in M$ satisfies the commitment, and $m' = m + g$ or $m' = m - g$ is such that $m' \in M$ we can replace m with m'. This can be repeated, and usually allows to find an element outside of the cone on the staircase image.

Remark however that multiple signatures of the same message represent in different ways the same element, hence leak a short lattice element. This means that the same document should never be signed twice. And allowing moving signatures has as a consequence that the private key is larger (at least twice as large) and the signing procedure is slower.

This has however a different positive side: the experience shows, not unexpectedly, that almost every signature falls inside a smaller box than M, for example $[0, 18]^n$, especially if we allow to move the signature as above, or if we can modify the document to sign. Reducing thus the commitment set of course increases the difficulty of signature forgery.

References

1. AM. Bigatti, R. LaScala, L. Robbiano, *Computing Toric Ideals.* J. Symb.Comp.**27** (1999), pp. 351–365.
2. M. Caboara, F. Caruso, C. Traverso. *Gröbner Bases in Public Key Cryptography.*, Proc. ISSAC '08, ACM (2008), pp. 315–323.
3. M. Caboara, F. Caruso, C. Traverso. *Lattice Polly Cracker cryptosystems*, accepted, J. Symb. Comp., 2010, currently available at `http://posso.dm.unipi.it/crypto`
4. O. Goldreich, S. Goldwasser, and S. Halevi. Public-key Cryptosystems from Lattice Reduction Problems. In *CRYPTO '97*, LNCS 1294, 112–131, 1997.
5. P. Nguyen. Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from Crypto '97. in *Crypto'99*, LNCS 1666 288–304, 1999.
6. P. Q. Nguyen, O. Regev Learning a Parallelepiped: Cryptanalysis of GGH and NTRU Signatures. *J. Cryptol.*, 22:139–160, 2009.

A public key exchange using semidirect products of groups

M. Habeeb, D. Kahrobaei and V. Shpilrain

1. Key Exchange Protocol

Recall the definition of a semidirect product:
Let H, Q be two groups, and let $\phi : Q \to Aut(H)$ be a homomorphism. Then the semidirect product of H and Q is $\Gamma = H \rtimes_\phi Q = \{(h, q) : h \in H, q \in Q\}$ where the group operation is given by

$$(h, q)(h', q') = (h\phi(q)(h'), qq').$$

Let now A and B be groups, $(b, a) \in B \times A$, and $n \in \mathbb{N}$. We require B to be an abelian group, with the additional condition that its automorphism group, $Aut(B)$, must contain a sufficiently large abelian subgroup. In our key exchange protocol, $A, B, Aut(B), (b, a)$, and n are public information.

Bob begins by choosing an embedding $\phi : A \to Aut(B)$, which he keeps secret. He then computes

$$x = (b, a)^n = (b\phi(a)(b)\phi(a^2)(b)\cdots\phi(a^{n-1})(b), a^n) \in B \rtimes_\phi A.$$

Bob then sends x to Alice.

Alice also chooses an embedding $\psi : A \to Aut(B)$, which she keeps secret, and computes

$$y = (b, a)^n = (b\psi(a)(b)\psi(a^2)(b)\cdots\psi(a^{n-1})(b), a^n) \in B \rtimes_\psi A.$$

Alice sends y to Bob.

Bob can find the element

$$\psi(a)(b)\psi(a^2)(b)\cdots\psi(a^{n-1})(b) \in B$$

by multiplying the first coordinate of y by b^{-1}, while Alice can find the element

$$\phi(a)(b)\phi(a^2)(b)\cdots\phi(a^{n-1})(b) \in B$$

by multiplying the first coordinate of x by b^{-1}. Denote $\phi(a)$ by ϕ_a and $\psi(a)$ by ψ_a.

Bob can now compute

$$\begin{aligned}\prod_{i=1}^{n-1} \phi_{a^i}(\psi_a(b)\psi_{a^2}(b)\cdots\psi_{a^{n-1}}(b)) &= \prod_{i=1}^{n-1} \phi_a^i(\psi_a(b)\psi_a^2(b)\cdots\psi_a^{n-1}(b)) \\ &= \prod_{i=1}^{n-1} \phi_a^i \circ \psi_a(b)\phi_a^i \circ \psi_a^2(b)\cdots\phi_a^i \circ \psi_a^{n-1}(b).\end{aligned}$$

Similarly, Alice can compute

$$\begin{aligned}\prod_{i=1}^{n-1} \psi_{a^i}(\phi_a(b)\phi_{a^2}(b)\cdots\phi_{a^{n-1}}(b)) &= \prod_{i=1}^{n-1} \psi_a^i(\phi_a(b)\phi_a^2(b)\cdots\phi_a^{n-1}(b)) \\ &= \prod_{i=1}^{n-1} \psi_a^i \circ \phi_a(b)\psi_a^i \circ \phi_a^2(b)\cdots\psi_a^i \circ \phi_a^{n-1}(b)).\end{aligned}$$

If in addition to B being abelian, we require that ϕ_a and ψ_a commute in $Aut(B)$, then we have the following equality:

$$\begin{aligned}\prod_{i=1}^{n-1} \phi_{a^i}(\psi_a(b)\psi_{a^2}(b)\cdots\psi_{a^{n-1}}(b)) &= \prod_{i=1}^{n-1} \phi_a^i(\psi_a(b)\psi_a^2(b)\cdots\psi_a^{n-1}(b)) \\ &= \prod_{i=1}^{n-1} \phi_a^i \circ \psi_a(b)\phi_a^i \circ \psi_a^2(b)\cdots\phi_a^i \circ \psi_a^{n-1}(b) \\ &= \prod_{i=1}^{n-1} \psi_a \circ \phi_a^i(b)\psi_a^2 \circ \phi_a^i(b)\cdots\psi_a^{n-1} \circ \phi_a^i(b) \\ &= \prod_{j=1}^{n-1} \psi_a^j \circ \phi_a(b)\psi_a^j \circ \phi_a^2(b)\cdots\psi_a^j \circ \phi_a^{n-1}(b)\end{aligned}$$

If we denote this element by k, then k is computable by both Bob and Alice as shown above, giving them a shared key. The security of this protocol relies on the difficulty of recovering the homomorphisms ϕ and ψ from $(b, a) \in B \times A$, x, and y. The most obvious way for an adversary to find the shared secret key is to determine either the homomorphism ϕ or ψ based on the public information. Since both x and y are public, an adversary can easily compute

$$\psi(a)(b)\psi(a^2)(b)\cdots\psi(a^{n-1})(b) \in B$$

and

$$\phi(a)(b)\phi(a^2)(b)\cdots\phi(a^{n-1})(b) \in B.$$

If the adversary determines ϕ, then he knows $\phi_a \in Aut(B)$ and can compute

$$\prod_{i=1}^{n-1} \phi_a^i(\psi_a(b)\psi_a^2(b)\cdots\psi_a^{n-1}(b)) = \prod_{i=1}^{n-1} \phi_a^i \circ \psi_a(b)\phi_a^i \circ \psi_a^2(b)\cdots\phi_a^i \circ \psi_a^{n-1}(b))$$

which is the shared key, k. Similarly, if the adversary determines ψ, he can also find the shared secret key.

2. Proposed Platform Group

The group B can be an additive abelian p-group of order p^m with the additional property that $pb = 0$ for every $b \in B$, where p is approximately 10^5. Then B may be considered an m-dimensional vector space over $\mathbb{Z}/p\mathbb{Z} = \mathbb{F}_p$. Under these conditions the group of automorphisms of B is $GL_m(\mathbb{F}_p)$. The order of $GL_m(\mathbb{F}_p)$ is $(p^m - 1)(p^m - p)\cdots(p^m - p^{m-1})$. Since we require a homomorphism $\phi : A \to Aut(B)$, the order of $\phi(a)$ must divide the order of a. We know that $Aut(B)$ has an element of order p, and so to ensure that we can find a non-trivial homomorphism ϕ, we require that the group A also be a p group of order p^l. Now $|B \rtimes_\phi A| = p^{m+l}$, and so $1 < n < p^{m+l}$. For security purposes, we require n to be sufficiently large.

If we require B and A to be as above, the homomorphisms ϕ_a and ψ_a have matrix representations J and K respectively, and the element b and the shared key k can be represented as column vectors.

We would like to ensure that it is easy to find homomorphisms ϕ_a, ψ_a that commute. We begin by choosing an arbitrary matrix $M \in GL_m(\mathbb{F}_p)$ and a natural number s with $s = \lfloor \frac{m}{2} \rfloor$, which are made public. Bob chooses an automorphism of the form $H = \begin{pmatrix} Q & 0 \\ 0 & I \end{pmatrix}$ and Alice chooses an automorphism of the form $S = \begin{pmatrix} I & 0 \\ 0 & R \end{pmatrix}$, where Q, R are $s \times s$ block matrices and I is the $(m-s)\times(m-s)$ identity matrix. The matrices H and S are kept secret. Bob and Alice select the matrices

$$J = P_B(MHM^{-1}) = \sum_{i=1}^{t'} c_i MH^iM^{-1}$$

and

$$K = P_A(MSM^{-1}) = \sum_{i=1}^{t} d_i MS^iM^{-1}$$

respectively, where $c_i, d_i \in \mathbb{F}_p$ are randomly selected coefficients. We would like to add the additional requirement that the matrices J and K do not have an "‘easy"’ form, such as diagonal, upper triangular or lower triangular matrices, although the probability of this occurring is negligible. The matrices J and K commute, and can be used as ϕ_a and ψ_a. Now computing the shared key takes the form

$$\begin{aligned}\sum_{i=1}^{n-1} J^i(K\cdot b + K^2\cdot b + \cdots K^{n-1}\cdot b) &= \sum_{i=1}^{n-1}(J^iK\cdot b + J^iK^2\cdot b + \cdots + J^iK^{n-1}\cdot b)\\ &= (J + J^2 + \cdots + J^{n-1})(K\cdot b + K^2\cdot b + \cdots + K^{n-1}\cdot b)\\ &= k\end{aligned}$$

or

$$\begin{aligned}\sum_{i=1}^{n-1} K^i(J\cdot b + J^2\cdot b + \cdots J^{n-1}\cdot b) &= \sum_{i=1}^{n-1}(K^iJ\cdot b + K^iJ^2\cdot b + \cdots K^iJ^{n-1}\cdot b)\\ &= (K + K^2 + \cdots + K^{n-1})(J\cdot b + J^2\cdot b + \cdots + J^{n-1}\cdot b)\\ &= k\end{aligned}$$

Finding the matrix K from the public information b and

$$K\cdot b + K^2\cdot b + \cdots + K^{n-1}\cdot b = g$$

is equivalent to solving the matrix equation

$$(K + K^2 + \cdots + K^{n-1})\cdot b = g. \qquad (2.1)$$

Similarly, from the public information we may find J by solving the equation

$$(J + J^2 + \cdots + J^{n-1})b = h. \qquad (2.2)$$

We may write these matrix equations as a system of m polynomial equations in the entries of K (or J). The search problem for polynomial equations over finite fields has been well studied and is known to be NP-hard [1]. By adding b to both sides of (2.1), which is public to the attacker, equation (2.1) becomes

$$(I + K + K^2 + \cdots + K^{n-1})\cdot b = g + b,$$

which is equivalent to

$$(I - K^n)\cdot b = (I - K)\cdot(g + b).$$

Although this simplifies equation (2.1) it does not simplify the problem at hand since we will still have a matrix equation, which can still be written as a system of non-linear equations over a finite field. We also note that if the matrices J and K are of a simple form (diagonal, upper or lower triangular) then this equation becomes easier to solve, but the probability of this occuring is neglible. The cryptosystem is secure against eigenvalue attacks as the matrices J and K are unknown.

References

[1] G. V. Bard. *Algebraic Cryptanalysis* (Springer-Verlag, 2009) 1-392

M. Habeeb
CUNY Graduate Center, City University of New York
e-mail: `MHabeeb@GC.Cuny.edu`

D. Kahrobaei
CUNY Graduate Center, City University of New York
e-mail: `DKahrobaei@GC.Cuny.edu`

V. Shpilrain
The City College of New York and CUNY Graduate Center
e-mail: `shpil@groups.sci.ccny.cuny.edu`

On lower bounds for Information Set Decoding over $\mathbb{F}_q$

Robert Niebuhr[1], Pierre-Louis Cayrel[2], Stanislav Bulygin[2], and Johannes Buchmann[1,2]

[1] Technische Universität Darmstadt
Fachbereich Informatik
Kryptographie und Computeralgebra,
Hochschulstraße 10
64289 Darmstadt
Germany
{rniebuhr, buchmann}@cdc.informatik.tu-darmstadt.de
[2] CASED – Center for Advanced Security Research Darmstadt,
Mornewegstrasse, 32
64293 Darmstadt
Germany
{pierre-louis.cayrel, stanislav.bulygin}@cased.de

Abstract. Code-based cryptosystems are promising candidates for post-quantum cryptography. The increasing number of cryptographic schemes that are based on codes over fields different from $\mathbb{F}_2$ requires an analysis of their security. Information Set Decoding (ISD) is one of the most important generic attacks against code-based cryptosystems. We give lower bounds for ISD over $\mathbb{F}_q$, thereby anticipating future software and hardware improvements. Our results allow to compute conservative parameters for cryptographic applications.

Key words: Information Set Decoding, lower bounds, codes, post quantum, cryptography.

Introduction

Error-correcting codes have been applied in cryptography for at least three decades since R. J. McEliece published his paper in 1978 [10]. It has received much attention as it is a promising candidate for post-quantum cryptography. McEliece used the class of binary Goppa codes for his construction, and most other schemes published since then have also been using binary codes.
However, in recent years, many new proposals use codes over larger fields $\mathbb{F}_q$, mostly in an attempt to reduce the size of the public and private keys. Two examples that received a lot of attention are quasi-cyclic codes [3] by Berger at al., and quasi-dyadic codes [11] (Misoczki-Barreto). The security, however, is not as well understand for q-ary codes as for binary ones: Faugère et al. [7] published an attack which broke these two cryptosystems for several sets of parameters.

This makes it important to analyze the complexity of attacks against code-based cryptosystems over larger fields $\mathbb{F}_q$.

The two most important types of attacks against code-based cryptosystems are structural attacks and decoding attacks. Structural attacks exploit structural weaknesses in the construction, and often they attempt to recover the private key. Decoding attacks are used to decode a given cipher text. In this paper, we will not consider structural attacks, since they are restricted to certain constructions or classes of codes. Information Set Decoding (ISD) is one of the most important generic decoding attacks, and it is the most efficient against many schemes.

Previous work

Over the years, there have been many improvements and generalizations of this attack, e.g. Lee-Brickell [9], Stern [14], Canteaut-Chabaud [6], Bernstein et al. [5]. Recently, two papers – Finiasz-Sendrier [8] and Peters [12] – studied this algorithm. The former provides lower bounds for the complexity of the ISD algorithm over $\mathbb{F}_2$, the latter describes how to generalize Stern's and Lee-Brickell's algorithms to $\mathbb{F}_q$.

Our contribution

In this paper, we propose and prove lower bounds for the complexity of ISD algorithms over $\mathbb{F}_q$. Our analysis gives an improvement of the efficiency of the ISD algorithm compared to Peters' analysis [12] and generalizes the lower bounds proposed by Finiasz and Sendrier in [8]. In addition to that, we show how to use the structure of $\mathbb{F}_q$ to increase the algorithm efficiency and compare our lower bounds with the ISD algorithm described by Peters. The details of the proof are given in the Appendix.

Organization of the paper

In Section 1, we start with a review of coding theory and cryptography over $\mathbb{F}_q$. The subsequent Section 2 presents the Information Set Decoding algorithm we are analyzing and states the lower bounds result. In Section 3, we apply these lower bounds to concrete parameters and compare the results with the most recent algorithm. We conclude in Section 4.

1 Review

1.1 Coding theory over $\mathbb{F}_q$

In general, a linear code $\mathcal{C}$ is a k-dimensional subspace of an n-dimensional vector space over a finite field $\mathbb{F}_q$, where k and n are positive integers with $k \leq n$, and q is a prime power. The error-correcting capability of such a code equals $(d-1)/2$, and it is the maximum number of errors that can be decoded. By (n, k, t), we denote a code that can efficiently decode $t \leq (d-1)/2$ errors. The co-dimension r of this code is defined by $r = n - k$.

Definition 1 (Hamming weight). *The (Hamming) weight* $\mathrm{wt}(x)$ *of a vector* x *is the number of its non-zero entries.*

Definition 2 (Minimum distance). *The (Hamming) distance* $d(x, y)$ *between two codewords* $x, y \in \mathcal{C}$ *is defined as the (Hamming) weight of* $x - y$*. The minimum weight* d *of a code* $\mathcal{C}$ *is defined as the minimum distance between any two different codewords, or equivalently as the minimum weight over all non-zero codewords:*

$$d := \min_{\substack{x,y \in \mathcal{C} \\ x \neq y}} d(x, y) = \min_{\substack{c \in \mathcal{C} \\ c \neq 0}} \mathrm{wt}(c).$$

A linear code of length n*, dimension* k *and minimum distance* d *is called an* $[n, k, d]$*-code.*

Definition 3 (Generator and Parity Check Matrix). *Let* $\mathcal{C}$ *be a linear code over* $\mathbb{F}_q$*. A generator matrix* G *of* $\mathcal{C}$ *is a matrix whose rows form a basis of* $\mathcal{C}$*:*

$$\mathcal{C} = \{xG : x \in \mathbb{F}_q^k\}.$$

A parity check matrix H *of* $\mathcal{C}$ *is defined by*

$$\mathcal{C} = \{x \in \mathbb{F}_q^n : Hx^T = 0\}$$

and generates the dual space of $\mathcal{C}$*. For a given parity check matrix* H *and any vector* e*, we call* s *the* syndrome *of* e *with* $s^T := He^T$ *.*

Remark 1. Two generator matrices generate *equivalent codes* if one is obtained from the other by a linear transformation or permutation. Therefore, we can write any generator matrix G in *systematic form* $G = [I_k|R]$, which allows a more compact representation. If $\mathcal{C}$ is generated by $G = [I_k|R]$, then a parity check matrix for $\mathcal{C}$ is $H = [-R^T|I_{n-k}]$ (up to permutation, H can be transformed so that the identity submatrix is on the left hand side).

The problems which cryptographic applications rely upon can have different numbers of solutions. For example, public key encryption schemes usually have exactly one solution, while digital signatures often have more than one possible solution. The uniqueness of solutions can be expressed by the Gilbert-Varshamov (GV) bound:

Definition 4 (q-ary Gilbert-Varshamov bound). *Let* $\mathcal{C}$ *be an* (n, k, t) *code over* $\mathbb{F}_q$*, and let* $r := n - k$*. The* q*-ary GV bound is the smallest integer* t_0 *such that*

$$\sum_{i=0}^{t_0} \binom{n}{i} (q-1)^i \geq q^r.$$

For large values of n, the last term dominates the sum, so the condition is often approximated by

$$\binom{n}{t_0}(q-1)^{t_0} \geq q^r.$$

If the number of errors that have to be corrected is smaller than the GV bound, then there is at most one solution. Otherwise, there can be several solutions.

1.2 The syndrome decoding problem and the McEliece PKC

Problem 1. Given a matrix H and a vector s, both over $\mathbb{F}_q$, and a non-negative integer t; find a vector $x \in \mathbb{F}_q^n$ of weight t such that $Hx^T = s^T$.

This problem was proved to be NP-complete in 1978 [4], but only for binary codes. In 1994, A. Barg proved that this result holds for codes over all finite fields ([1, in russian] and [2, Theorem 4.1].

Many code-based cryptographic schemes are based on the hardness of syndrome decoding. Among these are the McEliece cryptosystem and the CFS signature scheme. The latter, however, is unsuitable for q-ary codes, since it requires codes with a high density (ratio of the number of codewords to the cipher space size), and the density rapidly decreases with increasing field size q. We will therefore briefly describe the McEliece cryptosystem and show how it can be attacked by solving the syndrome decoding problem.

The McEliece PKC The McEliece public-key encryption scheme was presented by R. McEliece in 1978 ([10]). The original scheme uses binary Goppa codes, for which it remains unbroken, but the scheme can be used with any class of codes for which an efficient decoding algorithm is known.

Let G be a generator matrix for a linear (n, k, t)-code over $\mathbb{F}_q$, $\mathcal{D}_G$ a corresponding decoding algorithm. Let P be a $n \times n$ random permutation matrix and S an $k \times k$ invertible matrix over $\mathbb{F}_q$. These form the private key, while $(\widehat{G}, t)$ is made public, where $\widehat{G} = SGP$.

Encryption: Represent the plaintext as a vector m of length k over $\mathbb{F}_q$, choose a q-ary random error vector e of weight at most t, and compute the ciphertext

$$c = m\widehat{G} + e.$$

Decryption: Compute

$$\widehat{c} = cP^{-1} = mSG + eP^{-1}.$$

As P is a permutation matrix, eP^{-1} has the same weight as e. Therefore, $\mathcal{D}_G$ corrects these errors:

$$mSG = \mathcal{D}_G(\widehat{c})$$

Let $J \subseteq \{1, \dots, n\}$ be a set such that $G_{\cdot J}$ is invertible, then we can compute the plaintext

$$m = mSG \cdot G_{\cdot J}^{-1} \cdot S^{-1}.$$

Attacking the McEliece PKC Many variants of the McEliece encryption scheme have been proposed, often in an attempt to reduce the size of the public key. In most cases, these variants differ in which class of codes they use. Many of these variants have been broken, since a structural weakness had been found, but the original scheme using binary Goppa codes remains secure to date. For most parameters, ISD-like attacks are the most efficient attacks against the McEliece scheme (an exception is the CFS signature scheme, where a Generalized Birthday attack due to Bleichenbacher is more efficient).

2 Lower bounds for Information Set Decoding over $\mathbb{F}_q$

The algorithm we describe here recovers a q-ary error vector. It is a generalization of [8] to codes over $\mathbb{F}_q$. We first describe how to modify the algorithm to work over $\mathbb{F}_q$, then we show how to use the field structure to increase efficiency by a factor of $\sqrt{q-1}$.

In each step, we randomly re-arrange the columns of the parity check matrix H and transform it into the form

$$H = \left(\begin{array}{c|c} I_{n-k-l} & H_1 \\ 0 & H_2 \end{array} \right), \tag{1}$$

where I_{n-k-l} is the identity matrix of size $(n-k-l)$. Usually, the columns are chosen adaptively to guarantee the success of this step. Although this approach could bias the following steps, it has not shown any influence in practice. The variables l and p (see next step) are algorithm parameters optimized for each attack.

The error vector we are looking for has p errors in the column set corresponding to H_1 and H_2, and the remaining $(t-p)$ errors in the first $(n-k-l)$ columns. We first check all possible error patterns of p errors in the last $k+l$ columns such that the weighted sum S of those p columns equals the syndrome s in the last l rows. We do this by searching for collisions between the two sets L_1 and L_2 defined as

$$L_1 = \{H_2 e^T : e \in W_1\} \tag{2}$$

$$L_2 = \{s_2 - H_2 e^T : e \in W_2\}, \tag{3}$$

where $W_1 \subseteq \mathcal{W}_{k+l;\lfloor p/2 \rfloor;q}$ and $W_2 \subseteq \mathcal{W}_{k+l;\lceil p/2 \rceil;q}$ are given to the algorithm, and $\mathcal{W}_{k+l;p;q}$ is the set of all q-ary words of length $k+l$ and weight p. Writing

$e = [e'|e_1 + e_2]$ and $s = [s_1|s_2]$ with s_2 of length l, this means we search for vectors e_1 and e_2 of weight $\lfloor p/2 \rfloor$ and $\lceil p/2 \rceil$, respectively, such that

$$H_2 \cdot [e_1 + e_2]^T = s_2^T.$$

If this succeeds, we compute the difference $S - s$; if this does not have weight $t - p$, the algorithm restarts. Otherwise, the non-zero entries correspond to the remaining $t - p$ errors:

$$\begin{aligned} He^T &= \left(\begin{array}{c|c} I_{n-k-l} & H_1 \\ \hline 0 & H_2 \end{array}\right) \begin{pmatrix} e' \\ e_1 + e_2 \end{pmatrix} \\ &= \begin{pmatrix} I_{n-k-l} \cdot e'^T + H_1 \cdot (e_1 + e_2)^T \\ H_2 \cdot (e_1 + e_2)^T \end{pmatrix} \\ &= \begin{pmatrix} I_{n-k-l} \cdot e'^T \\ 0 \end{pmatrix} + S \\ &\overset{!}{=} \begin{pmatrix} s_1^T \\ s_2^T \end{pmatrix} \end{aligned}$$

Therefore, we have

$$I_{n-k-l} \cdot e'^T = s_1^T - H_1 \cdot (e_1 + e_2)^T,$$

revealing the remaining columns of e.

Using the field structure We can use the field structure of $\mathbb{F}_q$ to increase the algorithm efficiency. Note that for all vectors e such that $He^T = s^T$, there are $q - 1$ pairwise different vectors e' such that $He'^T = as^T$ for some $a \in \mathbb{F}_q \backslash \{0\}$, namely $e' = ae$. Clearly, if we find such an e', we can calculate e which solves the syndrome decoding problem. We can modify the algorithm to allow it to find these vectors e' as well, thereby increasing the fraction of error vectors that are (implicitly) tested in each iteration by a factor of $q - 1$ (see the Appendix for a detailed description).
Since this fraction is calculated using $|W_1| \cdot |W_2|$, we can also keep the fraction constant and decrease the size of the sets W_i by a factor of $\sqrt{q-1}$ each. As the work factor in each iteration of the algorithm is linear in $|W_1| + |W_2|$, this increases the algorithm efficiency by a factor of $\sqrt{q-1}$.

A simple way to decrease the size of the sets W_i is to redefine them as follows. For any vector a over $\mathbb{F}_q$, let us denote its first non-zero entry by $a(0) \in \mathbb{F}_q \backslash \{0\}$, and let

$$W_1' \subseteq \{e \in \mathcal{W}_{k+l;\lfloor p/2 \rfloor;q} : e(0) = 1\} \tag{4}$$

$$L_1' = \left\{(H_2 e^T)((H_2 e^T)(0))^{-1} : e \in W_1'\right\} \tag{5}$$

$$L_2' = \left\{(s_2 - H_2 e^T)((s_2 - H_2 e^T)(0))^{-1} : e \in W_2\right\}. \tag{6}$$

Remark 2. Note that even though the calculation of each vector is more costly due to the final division by the leading coefficient, this is by far offset by the smaller number of vectors that need to be calculated.

The algorithm thus works as follows:

Algorithm 1 Information Set Decoding over $\mathbb{F}_q$

```
Parameters:

 – Code parameters: Integers $n$, $r = n - k$ and $t$, and a finite field $\mathbb{F}_q$
 – Algorithm parameters: Two integers $p > 0$ and $l > 0$, and two sets $W_1 \subseteq \{e \in \mathcal{W}_{k+l;\lfloor p/2 \rfloor;q} : e(0) = 1\}$ and $W_2 \subseteq \mathcal{W}_{k+l;\lceil p/2 \rceil;q}$

Remark: The function $h_l(x)$ returns the last $l$ bits of the vector $x \in \mathbb{F}_q^n$. The variables $y := (He_1^T)(0)$ and $z := (s - He_2^T)(0)$ are notational shortcuts.
Input: Matrix $H_0 \in \mathbb{F}_q^{r \times n}$ and a vector $s_0 \in \mathbb{F}_q^r$
Repeat                                                        (MAIN LOOP)
  $P \leftarrow$ random $n \times n$ permutation matrix
  $(H, U) \leftarrow$ PGElim($H_0 P$)        //partial Gauss elimination as in (1)
  $s \leftarrow s_0 U^T$
  for all $e_1 \in W_1$
    $i \leftarrow h_l(He_1^T / y)$                                        (ISD 1)
    write($e_1, i$)                 //store $e$ in some data structure at index $i$
  for all $e_2 \in W_2$
    $i \leftarrow h_l((s_2^T - He_2^T)/z)$                                (ISD 2)
    $S \leftarrow$ read($i$)                 //extract the elements stored at index $i$
    for all $e_1 \in S$
      if wt($s^T - H(e_1 + e_2)^T) = t - p$                              (ISD 3)
        return $(P, e_1 z/y + e_2)$,                                    (SUCCESS)
```

Proposition 1. *If $\binom{n}{t}(q-1)^t < q^r$ (single solution), or if $\binom{n}{t}(q-1)^t \geq q^r$ (multiple solutions) and $\binom{r}{t-p}\binom{k}{p}(q-1)^t \ll q^r$, a lower bound for the expected cost (in binary operations) of the algorithm is*

$$WF_{qISD}(n, r, t, p, q) = \min_p \frac{1}{\sqrt{q-1}} \cdot \frac{2l \min\left(\binom{n}{t}(q-1)^t, q^r\right)}{\lambda_q \binom{r-l}{t-p}\binom{k+l}{p}(q-1)^t} \cdot \sqrt{\binom{k+l}{p}(q-1)^p}$$

with $l = \log_q\left(K_q \lambda_q \sqrt{\binom{k}{p}(q-1)^{p-1}} \cdot \ln(q)/2\right)$ and $\lambda_q = 1 - \exp(-1) \approx 0.63$. An exception is $p = 0$ where we cannot gain a factor of $\sqrt{q-1}$, hence

$$WF_{qISD}(n, r, t, 0, q) = \frac{\binom{n}{t}}{\binom{r}{t}}$$

If $\binom{n}{t}(q-1)^t \geq q^r$ and $\binom{r}{t-p}\binom{k}{p}(q-1)^t \geq q^r$, the expected cost is

$$WF_{qISD} \approx \min_{p} \frac{2lq^{r/2}}{\sqrt{\binom{r-l}{t-p}(q-1)^{t-p}}}$$

with $l \approx \log_q \left(K_{t-p} \frac{q^{r/2}}{\sqrt{\binom{r}{t-p}(q-1)^{t-p}}} \cdot \ln(q)/2 \right)$.

Remark 3. The variable K_q represents the number of operations required to check the condition (ISD 3). A realistic value for K_q is $K_q = 2t$, which will be used for the parameters in Section 3.

Remark 4. In the algorithm described above, all computations are done over $\mathbb{F}_q$, so the complexity also depends on the implementation of q-ary arithmetic. A naïve implementation yields an additional factor of $\log_2(q)$ for addition and $\log_2^2(q)$ for multiplication. There are several techniques to improve this, e.g. by lifting to $\mathbb{Z}[x]$ (for large q) or by precomputing exp and log tables (for small q). Especially for small q, this allows to make q-ary arithmetic nearly as fast as binary, so in order to gain conservative estimates, we will neglect this factor.

Remark 5. The total work factor is the product of the number of iterations by the work factor per iteration. In practice, the latter is essentially the sum of a matrix multiplication (with the permutation matrix), the Gaussian elimination, and the search for collisions between L_1' and L_2'. Compared with the binary case, the Gaussian elimination is slower in the q-ary case, because every row has to be divided by the pivot entry. However, since the matrix multiplication and the Gaussian elimination are much faster than the collision search, we do not allocate any cost to them.

3 Results

In [12], the author shows how to extend Lee-Brickell's and Stern's algorithms to codes over $\mathbb{F}_q$. The website [13] lists the work factor of this algorithm against several parameters. We use the same parameters and compare these results with our lower bound.

Table from C. Peters [13], containing parameters for quasi-cyclic [3] and quasi-dyadic [11] codes:

Code parameters				Claimed	$\log_2$(#bit ops)	Lower bound
q	n	k	w	security level	(from [13])	$\log_2$(#bit ops)
256	459	255	50	80	81.93	65.05
256	510	306	50	90	89.43	72.93
256	612	408	50	100	102.41	86.49
256	765	510	50	120	101.58	85.14
1024	450	225	56	80	83.89	62.81
1024	558	279	63	90	91.10	69.81
1024	744	372	54	110	81.01	58.39
4	2560	1536	128	128	181.86	173.23
16	1408	896	128	128	210.61	201.60
256	640	512	64	102	184.20	171.88
256	768	512	128	136	255.43	243.00
256	1024	512	256	168	331.25	318.61
2	2304	1281	64	80	83.38	76.86
2	3584	1536	128	112	112.17	105.34
2	4096	2048	128	128	136.47	129.05
2	7168	3073	256	192	215.91	206.91
2	8192	4096	256	256	265.01	254.16

For the algorithm from [12] as well as for our lower bound algorithm, the expected number of binary operations is the product of the number of iterations by the number of binary operations in each iteration. While the former factor is the same for both algorithms or even a little higher for our algorithm, the lower bound for the number of operations per iteration is much smaller in our case, which results in the difference between these algorithms.

The comparison below is between our algorithm and the overlapping-sets version from [12], since it is structurally closer to our algorithm than the even-split version. The runtime difference between these two versions is comparatively low.

3.1 Difference in the number of operations per iteration

The number of operations per iteration for the first algorithm is the sum of three steps:

1. Reusing parts of information sets and performing precomputations
2. Compute sums of p rows to calculate He^T
3. For each collision (e_1, e_2), check if $\text{wt}(s^T - H(e_1 + e_2)^T) = t - p$

To compare the cost of these steps with that used for our lower bound, we calculate all values for the $(450, 225, 56)$ parameter set over $\mathbb{F}_{1024}$. For this set, using $p = 1$, $l = 2$, $m = 1$, $c = 40$ and $r = 1$ (the last three are parameters specific for the first algorithm), we calculate a total cost of the first algorithm of $2^{76.6}$, which consists of 2^{52} iterations of $2^{24.6}$ operations each[3].

Precomputations The cost of the first step is given in [12] as

$$(n-1)\left((k-1)\left(1-\frac{1}{q^r}\right)+(q^r-r)\right)\frac{c}{r},$$

where c and r are algorithm parameters (i.e. r is *not* the co-dimension of the code). For these parameters, this amounts to $2^{24.4}$ operations, so it is the most expensive step.
Our algorithm does not use precomputation, so we allocate no cost.

Compute sums of $\boldsymbol{p}$ rows to calculate $\boldsymbol{He^T}$ The cost of this step for the first algorithm is

$$((k-p+1)+(N+N')(q-1)^p)\,l.$$

The parameters N and N' are the sizes of the sets and correspond to W_1 and W_2. For the parameters given above, this step adds $2^{19.3}$ operations.

Our algorithm allocates to this step a cost of

$$l|W_1'|+l|W_2| = 2l\sqrt{\binom{k+l}{p}(q-1)^{p-1}}.$$

We make this optimistic assumption[4] for the cost of a matrix-vector multiplication to anticipate further software and hardware improvements for this operation. The result is 2^6 operations in this case.

Check collisions The first algorithm allocates a cost of

$$\frac{q}{q-1}(w-2p)2p\left(1+\frac{q-2}{q-1}\right)\frac{NN'(q-1)^{2p}}{q^l}$$

to this step. For our set of parameters, this equals $2^{22.4}$ operations.

In our algorithm, we expect the number of collisions to be

$$\frac{\lambda_q|W_1'|\cdot|W_2|}{q^l}=\frac{\lambda_q\binom{k+l}{p}(q-1)^{p-1}}{q^l}.$$

The cost K_q to check each collision is taken to be $K_q=2t$. Since the expected number of collisions per iteration is very small, the expected cost per iteration

[3] The difference between this value and the one listed in the Table results from the fact the the latter were optimized with $p \geq 2$, while $p=1$ turns out to be better.
[4] From the cryptanalyst's point of view.

is < 1.

Some of the assumptions above may seem fairly optimistic. However, we find that necessary since we want to establish conservative lower bounds.

4 Conclusion and Outlook

In this paper, we have presented and proved lower bounds for Information Set Decoding algorithms over $\mathbb{F}_q$. Part of the result is a modification of the algorithms from [8] which allows to increase the efficiency of the algorithm by a factor of $\sqrt{q-1}$.

It can be seen from the table in Section 3 that over $\mathbb{F}_2$ the efficiency of concrete algorithms is not far from the lower bound, while over larger fields the gap is wider. We propose to further investigate improvements over $\mathbb{F}_q$ to decrease the size of this gap.

Also, in some situations an attacker has partial knowledge of the error vector. For example, in the FSB hash function it is known that the solution e (of $He^T = s^T$) is a regular word, that means that each block of size n/t has weight 1. It should be analyzed how partial knowledge of the solution can increase the efficiency of attacks in order to better estimate the security of cryptographic schemes.

References

[1] Barg, A.: Some New NP-Complete Coding Problems. In: *Probl. Peredachi Inf.* 30 (1994), S. 23–28. – (in Russian)

[2] Barg, A.: Complexity Issues in Coding Theory. In: *Electronic Colloquium on Computational Complexity (ECCC)* 4 (1997), Nr. 46

[3] Berger, T. P. ; Cayrel, P.-L. ; Gaborit, P. ; Otmani, A.: Reducing Key Length of the McEliece Cryptosystem. In: *AFRICACRYPT* Bd. 5580, Springer, 2009 (Lecture Notes in Computer Science), S. 77–97

[4] Berlekamp, E. ; McEliece, R. ; Tilborg, H. van: On the inherent intractability of certain coding problems. In: *IEEE Trans. Inform. Theory* 24 (1978), Nr. 3, S. 384–386

[5] Bernstein, D. J. ; Lange, T. ; Peters, C.: Attacking and defending the McEliece cryptosystem. In: *PQCrypto '08: Proceedings of the 2nd International Workshop on Post-Quantum Cryptography.* Berlin, Heidelberg : Springer-Verlag, 2008. – ISBN 978–3–540–88402–6, S. 31–46

[6] Canteaut, A. ; Chabaud, F.: A New Algorithm for Finding Minimum-Weight Words in a Linear Code: Application to McEliece's Cryptosystem and to Narrow-Sense BCH Codes of Length 511. In: *IEEE Transactions on Information Theory* 44 (1998), Nr. 1, S. 367–378

[7] Faugère, J.-C. ; Otmani, A. ; Perret, L. ; Tillich, J.-P.: *Algebraic Cryptanalysis of McEliece Variants with Compact Keys.* 2009. – (preprint)

[8] Finiasz, M. ; Sendrier, N.: Security Bounds for the Design of Code-based Cryptosystems. In: *Advances in Cryptology – Asiacrypt'2009*, 2009. – http://eprint.iacr.org/2009/414.pdf

[9] Lee, P.J. ; Brickell, E.F.: An observation on the security of McEliece's public-key cryptosystem. In: *EUROCRYPT '88, Lect. Notes in CS*, 1988, S. 275–280

[10] McEliece, R.J.: A Public-key cryptosystem based on algebraic coding theory. In: *DNS Progress Report* (1978), S. 114–116

[11] Misoczki, R. ; Barreto, P. S. L. M.: Compact McEliece Keys from Goppa Codes. In: *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009* Bd. 5867, Springer, 2009 (Lecture Notes in Computer Science)

[12] Peters, C.: *Information-set decoding for linear codes over* $\mathbb{F}_q$. Cryptology ePrint Archive, Report 2009/589, 2009. – http://eprint.iacr.org/

[13] Peters, C.: *Iteration and operation count for information-set decoding over* $\mathbb{F}_q$. Jan 2010. – `http://www.win.tue.nl/~cpeters/isdfq.html`

[14] Stern, J.: A method for finding codewords of small weight. In: *Proc. of Coding Theory and Applications*, 1989, S. 106–113

A Proof of Proposition 1

Except for the additional factor of $1/\sqrt{q-1}$, the proof is similar to that in [8]. We will use the same approach and focus on the differences. As above, let $y(0)$ denote the first non-zero entry of vector $y \in \mathbb{F}_q^n \backslash \{0\}$.

A.1 Efficiency improvement using the field structure of $\mathbb{F}_q$

The step of the algorithm that can be made more efficient using the field structure of $\mathbb{F}_q$ is the search for a pair (e_1, e_2) such that $e_1 \in \mathcal{W}_{k+l;\lfloor p/2\rfloor;q}$, $e_2 \in \mathcal{W}_{k+l;\lceil p/2\rceil;q}$ and

$$He_1^T = s_2^T - He_2^T,$$

where $\mathcal{W}_{k+l;p;q}$ is the set of all q-ary words of length $k+l$ and weight p.

Let W_1', W_2, L_1' and L_2' be defined as in (4)-(6). First note that for any pair (e_1, e_2) and all non-zero values $y \in \mathbb{F}_q$, we have

$$He_1^T = s_2^T - He_2^T \Leftrightarrow (He_1^T)y^{-1} = (s_2^T - He_2^T)y^{-1}.$$

Instead of He_1^T and $s_2^T - He_2^T$, we can store $(He_1^T)(He_1^T(0))^{-1}$ in L_1' and $(s_2^T - He_2^T)((s_2^T - He_2^T)(0))^{-1}$ in L_2', respectively. The list L_1', however, would contain every entry exactly $(q-1)$ times, since for every $y \in \mathbb{F}_q \backslash \{0\}$, e_1 and ye_1 yield the same entry. Therefore, we can generate the first list using only vectors e_1 whose first non-zero entry is 1.

To see that there is exactly one collision between L_1' and L_2' for every solution of the problem, let (e_1, e_2) be a pair found by our algorithm. Let $y = He_1^T(0)$ and $z = (s^T - He_2^T)(0)$. Then we have

$$(He_1^T)y^{-1} = (s^T - He_2^T)z^{-1},$$

and therefore $(e_1 z y^{-1}, e_2)$ is a solution to the problem.

Conversely, let (e_1, e_2) be a solution to the problem, i.e. $He_1^T + He_2^T = s_2^T$. We want to show that there exists a collision between L'_1 and L'_2 which corresponds to this solution. Let $y = He_1^T(0)$ and $z = (s_2^T - He_2^T)(0)$. Since $He_1^T = s_2^T - He_2^T$, we have

$$(He_1^T)y^{-1} = (s_2^T - He_2^T)z^{-1}. \tag{7}$$

As we did not limit the set W_2, the right hand side of equation (7) belongs to L'_2.

Let $x = e_1(0)$. The first non-zero entry of $e'_1 = e_1 x^{-1}$ is 1, so it was used to calculate one member of L'_1. As $He_1'^T(0) = (H(e_1 x^{-1})^T)(0) = yx^{-1}$,

$$(He_1'^T)((He_1'^T)(0))^{-1} = (H(e_1 x^{-1})^T)(yx^{-1})^{-1} = (He_1^T)y^{-1}.$$

Therefore, the left hand side of equation (7) belongs to L'_1.
Since $z = y$, this collision between L'_1 and L'_2 corresponds to the solution (e_1, e_2).

Obviously, this improvement can only be applied if $p > 0$, i.e. if there actually is a search for collisions. If $p = 0$, we are simply trying to find a permutation which shifts all error positions into the first r positions of s, so the runtime is the inverse of the probability P_0 of this event with $P_0 = \binom{r}{t}/\binom{n}{t}$. For the rest of this section we assume $p > 0$.

A.2 Cost of the algorithm

In most cases, the value of t will be smaller than the GV bound, and we expect the algorithm to require many iterations. In that case, in one iteration of our Main Loop, we expect to test a fraction $\lambda_q(z) = 1 - \exp(z_q)$ of vectors in $\mathcal{W}_{k+l;p;q}$, where

$$z_q = \frac{|W'_1| \cdot |W_2|}{\binom{k+l}{p}(q-1)^{p-1}}. \tag{8}$$

The success probability of each pair (e_1, e_2) is the number of pairs matching the syndrome in the last l rows, divided by the total number of possible values of He with $e \in \mathcal{W}_{k+l;p;q}$. Depending on the code parameters, the latter is either given by the number of error patterns or by the number of syndromes:

$$P_q = \frac{\lambda_q(z_q)\binom{r-l}{t-p}(q-1)^{t-p}}{\min\left(\binom{n}{t}(q-1)^t, q^r\right)}.$$

The success probability in one iteration of Main Loop is hence:

$$\begin{aligned} P_{p;q}(l) &= 1 - (1 - P_q)^{\binom{k+l}{p}(q-1)^p} \\ &\approx 1 - \exp(-P_q \cdot \binom{k+l}{p}(q-1)^p) \\ &= 1 - \exp\left(-\frac{\lambda_q(z_q)}{N_{p;q}(l)}\right), \end{aligned}$$

where

$$N_{p;q}(l) = \frac{\min\left(\binom{n}{t}(q-1)^t, q^r\right)}{\binom{r-l}{t-p}\binom{k+l}{p}(q-1)^t}.$$

For small $P_{p;q}(l)$, the cost of the algorithm can be calculated approximately as

$$\frac{N_{p;q}(l)}{\lambda_q(z_q)} \cdot \left(l|W_1'| + l|W_2| + K_q \frac{\lambda_q(z_q)\binom{k+l}{p}(q-1)^{p-1}}{q^l} \right),$$

which is the approximate number of iterations times the number of operations per iteration. K_q is the expected cost to perform the check $\mathrm{wt}(s^T - H(e_1+e_2)^T) = t-p$.

It is easy to see that we minimize this formula by choosing $|W_1'| = |W_2|$,

$$N_{p;q}(l) \cdot \left(2l \frac{|W_1'|}{\lambda_q(z_q)} + K_q \frac{\binom{k+l}{p}(q-1)^{p-1}}{q^l} \right).$$

Using (8), we get

$$N_{p;q}(l) \cdot \left(2l \frac{\sqrt{z_q}}{\lambda_q(z_q)} \sqrt{\binom{k+l}{p}(q-1)^{p-1}} + K_q \frac{\binom{k+l}{p}(q-1)^{p-1}}{q^l} \right).$$

Analytically, the optimal value for z_q is $z \approx 1.25$, but $z_q = 1$ is very close to optimal. Hence we choose $z_q = 1$, set $\lambda_q = \lambda_q(1) = 1 - e^{-1}$ and use (8),

$$N_{p;q}(l) \sqrt{\binom{k+l}{p}(q-1)^{p-1}} \frac{2}{\lambda_q} \cdot \left(l + \frac{K_q\lambda_q}{2} \cdot \frac{\sqrt{\binom{k+l}{p}(q-1)^{p-1}}}{q^l} \right).$$

The optimal value for l can be approximated by $l = \log_q\left(K_q\lambda_q\sqrt{\binom{k+l}{p}(q-1)^{p-1}} \cdot \ln(q)/2\right)$. In practice, we use $l \approx \log_q\left(K_q\lambda_q\sqrt{\binom{k}{p}(q-1)^{p-1}} \cdot \ln(q)/2\right)$. For small values of q, the factor $(\ln(q)/2)$ can be neglected. Hence the cost is

$$\frac{1}{\sqrt{q-1}} \cdot \frac{2l\min\left(\binom{n}{t}(q-1)^t, q^r\right)}{\lambda_q\binom{r-l}{t-p}\binom{k+l}{p}(q-1)^t} \cdot \sqrt{\binom{k+l}{p}(q-1)^p}.$$

Minimizing over p gives the result.

Now consider the case where $\binom{r}{t-p}\binom{k}{p}(q-1)^t \geq q^r$. Then the Main Loop is likely to succeed after a single iteration. This corresponds to the birthday algorithm described in [8]:

$$\mathrm{WF}_{\mathrm{BA}} \approx \frac{2}{\sqrt{P}} \cdot \left(l + \frac{K_0}{2\sqrt{P}2^l} \right).$$

We can apply this result here, since it does not depend on the field size, but only on the success probability. In the q-ary case this formula becomes

$$\mathrm{WF}_{\mathrm{qBA}} \approx \frac{2}{\sqrt{P}} \cdot \left(l + \frac{K_0}{2\sqrt{P}q^l} \right).$$

Easy analysis shows that the optimal value for l is

$$l = \log_q \left(\frac{\ln(q) K_0}{2\sqrt{P}} \right).$$

Applying this in our case with K_{t-p} instead of K_0 (since K_0 is the cost of the third step in the algorithm of [8], which is K_{t-p} when applied in the case of ISD), using

$$P = P_q \approx \frac{\binom{r-l}{t-p}(q-1)^{t-p}}{q^r},$$

and minimizing over p yields the lower bound result:

$$\mathrm{WF}_{\mathrm{qISD}} \approx \frac{2lq^{r/2}}{\sqrt{\binom{r-l}{t-p}(q-1)^{t-p}}}$$

with $l \approx \log_q \left(K_{t-p} \frac{q^{r/2}}{\sqrt{\binom{r-l}{t-p}(q-1)^{t-p}}} \cdot \ln(q)/2 \right)$.

On homogeneous polynomial decomposition

Paula Bustillo and Jaime Gutierrez[1]

Universidad de Cantabria, Santander, Spain

Extended Abstract

In [9], the authors proposed a new cryptosystem called 2R-scheme inspired by the C^*-cryptosystem, see [7]. In a 2R-scheme the space of plain texts and ciphertexts is $\mathbb{F}_q^m$, where $\mathbb{F}_q$ is a finite field of q elements. The secret key items are three affine bijections $r, s, t : \mathbb{F}_q^m \longrightarrow \mathbb{F}_q^m$ and two applications $\phi, \psi : \mathbb{F}_q^m \longrightarrow \mathbb{F}_q^m$ given by m quadratic equations over $\mathbb{F}_q$. The public key is the polynomial representation of the application $t \circ \psi \circ s \circ \phi \circ r : \mathbb{F}_q^m \longrightarrow \mathbb{F}_q^m$. This representation consists of m polynomials of degree 4.

The above applications ϕ and ψ are chosen among easily invertible ones in order to make decryption easy. For all proposed easily invertible applications at that time, the one-round schemes were broken, i.e., the analogous cryptosystems with secret key $s \circ \phi \circ r$. Therefore, the security of 2R-schemes was based on the difficulty of decomposing a list of m polynomials in $\mathbb{K}[\mathbf{x}] = \mathbb{K}[x_1, \ldots, x_m]$, where $\mathbb{K}$ is an arbitrary field. The paper [10] proposed efficient attacks that make the system insecure if m or $m-1$ polynomials in the list are given. Inspired by these ideas, in [1], the authors presented and algorithm that given a list $\mathbf{f} = (f_1, \ldots, f_u)$ of u homogeneous polynomials of degree 4 in m variables, finds lists $\mathbf{g} = (g_1, \ldots, g_u)$ and $\mathbf{h} = (h_1, \ldots, h_m)$ of homogeneous polynomials of degree 2 in m variables such that $f_i = g_i(h_1, \ldots, h_m)$ for all $i \in \{1, \ldots, u\}$, under some favourable circumstances. The algorithm was extended in [3] to a list of polynomials $\mathbf{f}$ of arbitrary degree $n = r \cdot s$. There is an improvement of the algorithm in [2], together with an algorithm for a list $\mathbf{f}$ of polynomials of degrees $r_1, \ldots, r_u$ respectively such that $s > 1$ divides all degrees.

Computation of intermediate $\mathbb{K}$-algebras and (r, s)-decompositions

We aim here at finding the relation among the concept of (r, s)-decomposition of homogeneous polynomials proposed in [1] and the computation of intermediate $\mathbb{K}$-algebras and intermediate fields We shall start by the definition of (r, s)-decomposable polynomials:

Definition 1. *Let $\mathbf{f} = (f_1, \ldots, f_u) \in \mathbb{K}[\mathbf{x}]^u$ be a list of homogeneous polynomials of degree $n = rs$. We say that $\mathbf{f}$ is (r, s)-decomposable if there exist a list $\mathbf{g} = (g_1, \ldots, g_u) \in \mathbb{K}[\mathbf{x}]^u$ of homogeneous polynomials of degree r and a list $\mathbf{h} = (h_1, \ldots, h_m) \in \mathbb{K}[\mathbf{x}]^m$ of homogeneous polynomials of degree s such that $f_i = g_i(h_1, \ldots, h_m)$, written $\mathbf{f} = \mathbf{g} \circ \mathbf{h}$. The tuple $(\mathbf{g}, \mathbf{h})$ is called an (r, s)-decomposition of $\mathbf{f}$.*

If A is a regular matrix, then $\mathbf{g} \circ \mathbf{h} = \mathbf{g} \circ A^{-1} \circ A \circ \mathbf{h}$. To avoid this ambiguity, two decompositions $(\mathbf{g}, \mathbf{h})$ and $(\mathbf{g}', \mathbf{h}')$ of a polynomial are defined to be **equivalent** if there exists a regular matrix A such that $\mathbf{h}'^T = A\mathbf{h}^T$. By this equivalence relation, we guarantee that two non-equivalent decompositions provide two different intermediate $\mathbb{K}$-algebras.

It is easy to see that $\mathbf{f}$ has an (r, s)-decomposition $(\mathbf{g}, \mathbf{h})$ if and only if $\mathbb{K}[\mathbf{f}] \subset \mathbb{K}[\mathbf{h}]$. Moreover, this relation is bijective:

Proposition 1. *Non-equivalent (r, s)-decompositions of a list of polynomials $\mathbf{f} = (f_1, \ldots, f_u)$ correspond bijectively to $\mathbb{K}$-algebras in $\mathbb{K}[\mathbf{f}] \subset \mathbb{K}[\mathbf{x}]$ generated by m homogeneous polynomials of degree s.*

This bijective relation does not extend to a bijective relation among the (r, s)-decompositions of $\mathbf{f}$ and the proper fields in $\mathbb{K}(\mathbf{f}) \subset \mathbb{K}(\mathbf{x})$ generated by a list $\mathbf{h}$ of homogeneous polynomials of degree s in general.

The algoritm of Faugère and Perret only finds an (r, s)-decomposition of $\mathbf{f}$ if $\mathbf{f}$ has only one non-equivalent decomposicion, i.e., it only finds a decomposition when there is exactly one intermediate $\mathbb{K}$-algebra (field) in $\mathbb{K}[\mathbf{f}] \subset \mathbb{K}[\mathbf{x}]$ (in $\mathbb{K}(\mathbf{f}) \subset \mathbb{K}(\mathbf{x})$) generated by m homogeneous polynomials of degree s.

The dimension of (r, s)-decomposable polynomials

In [5], the dimension of the decomposable univariate polynomials over an algebraically closed field is counted, and in [4], the author counts the dimension of the so called uni-multivariate decomposable polynomials, see [6], over an algebraically closed field. Here, we try counting the dimension of (r, s)-decomposable polynomials in m variables over an algebraically closed field.

From now on, $\mathbb{K}$ will denote an algebraically closed field. Let $P_{m,n} = \{f \in \mathbb{K}[\mathbf{x}] : f \text{ is homogeneous of degree } n\}$ be the vector space of homogeneous polynomials of degree n in m variables of dimension $a_{m,n} = \binom{m+n-1}{n}$

By arranging the monomials of degree n in m variables with respect to the lexicographical order $>_{lex}$, $m(1) = x_1^n, m(2) = x_1^{n-1}x_2, \ldots, m(a_{m,n}) = x_m^n$, we can identify a polynomial in $P_{m,n}$ sorted with respect to the lexicographical order with a tuple in $\mathbb{K}^{a_{m,n}}$, thus identifiying $P_{m,n}$ with the affine space $\mathbb{K}^{a_{m,n}}$.

For $n = rs$, we have the composition map

$$\begin{array}{rccc} \gamma_{m,n,r} : & P_{m,r} \times {P_{m,s}}^m & \longrightarrow & P_{m,n} \\ & (g, h_1, \ldots, h_m) & \mapsto & g(h_1, \ldots, h_m) \end{array}$$

Clearly, the set $D_{m,n,r}$ of (r, s)-decomposable polynomials of degree n is Im $\gamma_{m,n,r}$. The map $\gamma_{m,n,r}$ can be identified with a polynomial map

$$\Gamma_{m,n,r} : \mathbb{K}^{a_{m,r}} \times (\mathbb{K}^{a_{m,s}})^m \longrightarrow \mathbb{K}^{a_{m,n}}$$

that sends the coefficients of $g, h_1, \ldots, h_m$ to the coefficients of $g(h_1, \ldots, h_m)$. This map identifies $D_{m,n,r}$ with $\mathrm{Dec}_{m,n,r} = \mathrm{Im}\, \Gamma_{m,n,r}$. We aim at finding the dimension of the Zariski closure of $\mathrm{Dec}_{m,n,r}$, $\overline{\mathrm{Dec}_{m,n,r}}$.

A straightforward way to compute the dimension is to combine a suitable normalization in (r, s)-decomponsitions with the following theorem:

Theorem 1. ([8]) *Let X, Y be algebraic sets over $\mathbb{K}$. If $f : X \longrightarrow Y$ is a dominating polynomial map, i.e., such that $Y = \overline{f(X)}$, then there exists an open subset U in Y such that $f^{-1}(y)$ has dimension* $\dim X - \dim Y$ *for all $y \in U$.*

As a consequence, if the map $\Gamma_{m,n,r}|_X : X \longrightarrow \overline{\mathrm{Dec}_{m,n,r}}$ is dominating and such that all polynomials in $\mathrm{Dec}_{m,n,r} \setminus C$ have a finite number of (r,s)-decompositions, for a closed set $C \neq \mathrm{Dec}_{m,n,r}$, then $\dim \mathrm{Dec}_{m,n,r} = \dim X$.

It is clear that for $X = \mathbb{K}^{a_{m,r}+m\cdot a_{m,s}}$ the hypothesis are not satisfied: whenever a polynomial f has the (r,s)-decomposition $f = g \circ \mathbf{h}$, we can decompose f as $f = (g \circ A^{-1}) \circ (A \circ \mathbf{h})$ for every $A \in \mathrm{GL}_m(\mathbb{K})$.

Assume that $f = g(h_1, \ldots, h_m)$ is an (r,s)-decomposition of f where $h_1, \ldots, h_m$ are linearly independent. Then, the vector space generated by $h_1, \ldots, h_m$ is also generated by m homogeneous polynomials $h'_1, \ldots, h'_m$ of degree s such that each polynomial is monic with respect to the lexicographical order, $\mathrm{lm}(h'_1) >_{lex} >_{lex} \ldots >_{lex} \mathrm{lm}(h'_m)$, and $\mathrm{coeff}_{\mathrm{lm}(h_i)}(h_j) = 0$ for $i \neq j$, where $\mathrm{lm}(t)$ denotes the leading monomial of the polynomial t and $\mathrm{coeff}_m(t)$ is the coefficient of the monomial m in the polynomial t. Then, for $\mathbf{h}' = (h'_1, \ldots, h'_m)$, there exists an homogenous polynomial g' in m variables of degree r such that $f = g' \circ \mathbf{h}'$.

Let $V(i_1, \ldots, i_m)$ be the set of vector spaces generated by m polynomials $h_1, \ldots, h_m$, where $i_1 < i_2 < \cdots < i_m$, each h_j is monic with leading coefficient $m(i_j)$, and $\mathrm{coeff}_{\mathrm{lm}(h_j)}(h_i) = 0$ if $i \neq j$:

$$\begin{array}{c} \\ h_1 \rightarrow \\ h_2 \rightarrow \\ \\ h_m \rightarrow \end{array} \begin{array}{c} \begin{array}{cccccccc} & i_1 & & i_2 & & & i_m & \end{array} \\ \begin{pmatrix} 0 & 1 & \cdots & 0 & \cdots & \cdots & 0 & \cdots \\ 0 & 0 & 0 & 1 & \cdots & \cdots & 0 & \cdots \\ 0 & 0 & 0 & 0 & \cdots & \cdots & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \cdots \end{pmatrix} \end{array}$$

Each vector space in $V(i_1, \ldots, i_m)$ can be determined by $m \cdot (a_{m,s} - m)$ coefficients in $\mathbb{K}$ at most, thus identifying $V(i_1, \ldots, i_m)$ with $\mathbb{K}^{m\cdot(a_{m,s}-m)}$.

Let $\hat{V} = \cup_{1 \le i_1 < i_2 < \ldots < i_m \le a_{m,s}} V(i_1, \ldots, i_m)$ and V be the algebraic set corresponding to $\hat{V}$ by the identification between $P_{m,s}$ and $\mathbb{K}^{a_{m,s}}$. Then, $Dec_{m,n,r} = \mathrm{Im}\ \Gamma_{m,n,r}(\mathbb{K}^{a_{m,r}} \times \hat{V})$. Clearly, $\dim \overline{\mathrm{Dec}_{m,n,r}} = \dim \overline{\mathrm{Im}\ \Gamma(\mathbb{K}^{a_{m,r}} \times V)} \le a_{m,r} + m\cdot(a_{m,s} - m)$. Therefore, if it could be proven that $\dim \overline{\mathrm{Im}\ \Gamma(\mathbb{K}^{a_{m,r}} \times V(1,2,\ldots,m))} = a_{m,r} + m \cdot (a_{m,s} - m)$, then $\dim \overline{\mathrm{Dec}_{m,n,r}} = a_{m,r} + m \cdot (a_{m,s} - m)$.

For $(2,2)$-decompositions in two variables it can be proven that $\dim \overline{\mathrm{Dec}_{2,4,2}} = 3 + 2(3-2) = 5$ by using Gröbner basis computations.

Counting the dimension of decomposable lists of homogeneous polynomials of the same degree is completely analogous. Let $\mathrm{Dec}_{m,n,r,u}$ be the set of lists $\mathbf{f}$ of u homogeneous polynomials in $\mathbb{K}[\mathbf{x}]$ of degree n that are (r,s)-decomposable, and let

$$\Gamma_{m,n,r,u} : (\mathbb{K}^{a_{m,r}})^u \times V \longrightarrow \overline{\mathrm{Dec}_{m,n,r,u}}$$

be the function that maps the coefficients of the normalized tuple $(\mathbf{g}, \mathbf{h})$ to the coefficients of $\mathbf{g} \circ \mathbf{h}$. If the above normalization were the good one, then the dimension of $\overline{\mathrm{Dec}_{m,n,r,u}}$ would be $\dim((\mathbb{K}^{a_{m,r}})^u \times V) = u \cdot a_{m,r} + m \cdot (a_{m,s} - m)$.

References

1. Faugère, J.-C., Perret, L.: Cryptanalysis of $2R^-$ schemes. Advances in cryptology—CRYPTO 2006. Lecture Notes in Comput. Sci. **4117** (2006) 357–372
2. Faugère, J.-C., Perret, L.: High order derivatives and decomposition of multivariate polynomials. ISSAC '09: Proceedings of the 2009 international symposium on Symbolic and algebraic computation (2009) 207–214
3. Faugère, J.-C., Perret, L.: An efficient algorithm for decomposing multivariate polynomials and its applications to cryptography. Journal of Symbolic Computation **44** (2009) 1676–1689
4. von zur Gathen, J.: Counting decomposable multivariate polynomials. Technical Report arXiv:0811.4726 (2008)
5. von zur Gathen, J.: The number of decomposable univariate polynomials. ISSAC '09: Proceedings of the 2009 international symposium on Symbolic and algebraic computation (2009) 359–366
6. von zur Gathen, J., Gutierrez, J. Rubio, R.: Multivariate polynomial decomposition. Applicable Algebra in Engineering, Communication and Computing **14**, (2003) 11–31,
7. Matsumoto, T., Imai, H.: Public quadratic polynomial-tuples for efficient signature-verification and message-encryption. Advances in cryptology—EUROCRYPT '88. Lecture Notes in Comput. Sci. **330** (1988) 419–453
8. Mumford, D.: The red book of varieties and schemes. Lecture Notes in Mathematics **1358**, Springer-Verlag (1988)
9. Patarin, J., Goubin, L.: Asymmetric Cryptography wiht S-Boxes. Proceedings of ICICS'97, Lecture Notes in Comput. Sci. **1334** (1997) 369–380
10. Ye, D., Dai, Z., Lam, K.-Y.: Decomposing attacks on asymmetric cryptography based on mapping compositions. Journal of Cryptology. The Journal of the International Association for Cryptologic Research **14**, (2001) 137–150

An Efficient Method for Deciding Polynomial Equivalence Classes

Tianze Wang[1,2] and Dongdai Lin[1]

[1] SKLOIS, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
[2] Graduate University of Chinese Academy of Sciences, Beijing 100149, China
wtziscas@hotmail.com, ddlin@is.iscas.ac.cn

Abstract. The enumeration of isomorphism of polynomials (IP) problem is first introduced in [1], which consists of counting the number of solutions for an instance of IP problem and counting the number of different equivalence classes. In this paper we give a method to find all the polynomial equivalence classes under the IP with one secret problem for the cases of even characteristic ground field except F_2. We fist identify a rough classification according to matrix equivalence relation then in each matrix equivalence class we get finer classification according to the linearly equivalence relation. The given method is much more effective than exhaustive search algorithm and feasible for small n and q.

Keywords: Enumerative Problem, Isomorphism of Polynomials, Equivalence Class, Equivalent Keys

1 Introduction

Multivariate public key schemes dates back to the mid eighties with the design of MI scheme [2], from then on there are many papers on this direction. The security is based on the problem of solving system of nonlinear multivariate equations over a finite field which was proven to be NP-hard [3]. Since multivariate public key cryptography is proposed as the alternative to RSA cryptosystem and there is no quantum algorithms for that hard problem, it is of our interest.

In multivariate public key schemes it is usual to hide an easily inverted multivariate polynomial system $\mathbf{a}$ by composing two invertible affine transformations, say S and T, then the resulting polynomial system $\mathbf{b} = T \circ \mathbf{a} \circ S$ is random-looking. And this is highly related to another hard and fundamental problem, namely the isomorphism of polynomials (IP) problem. The IP problem is recovering the secret transformations S and T given $\mathbf{a}$ and $\mathbf{b}$.

Recently Lin et. al. introduced the corresponding enumerative problem of IP problem in [1]. This problem has two meanings: one is to identify the number of solutions of IP problem, which is equivalent to compute the number of equivalent keys for a fixed system of polynomials as the central function of one scheme. As we know, the isomorphism of polynomials can induce an equivalence relation, hence we can get a partition of all polynomial systems according to this equivalence relation. Thus the other meaning is to identify all the equivalence classes. Obviously not all system of nonlinear equations are hard to solve, and intuitively we think that the polynomial systems in the equivalence class containing some easy instance are also easily solved. Therefore we should avoid to use those instances.

Related works. The overwhelming majority of previous works are dedicated to find a solution of instances of IP with two secrets and IP with one secret problems, however they all neglect to consider the problem of identifying the number of the solutions of that problem which is related to the equivalent keys for a fixed central function and finding all equivalence classes.

In [4], the authors for the first time considered the equivalent keys of some multivariate public key schemes, such as C^*, HFE and oil-vinegar schemes. And they introduced some sustaining transformations, in which the "Big sustainer" and "Frobenius sustainer" are used to analyze the SFLASH [5, 6] and subfield variant of HFE schemes [7]. However they did not consider the general case, that is they did not connect the problem of equivalent key with the polynomial isomorphism problem which is a fundamental hard problem in MPKC.

In [1], the authors introduced a new tool, namely finite geometry, to study the enumerative problem of IP. And they gave some lower-bounds of the number of IP classes. Then they applied this new tool on an generalized MPKC scheme, i.e. "MI-like" shceme, they got the conclusion that there are many "MI-like" instances in HFE schemes which are insecure.

Our results. As we know, there is no algorithm for identifying all linearly equivalence classes from the set of all multivariate homogeneous quadratic polynomial systems. The interests of the researchers are of IP problem. But in this paper, we focus on the enumerative problem of equivalence classes and we will show how to find the complete classification of polynomials according to the classification by friendly mapping Ψ_1. We give an heuristic algorithm to determine the number of equivalence classes under the problem of IP with one secret. This method that underlies the algorithms takes advantage the invariant properties of the "diagonal" polynomials under the actions of linear transformations.

We first define a equivalence relation, i.e. matrix equivalence, which is the necessary condition of linearly equivalence. Thus we get the rough classification according to the matrix equivalence relation. Then based on the rough classification and the stabilizer computed already we can get the finer classes, i.e. the linearly equivalence classes. Empirically the orders of the stabilizers are much less than the order of general linear group. Thus the efficiency of the given algorithm is higher than exhaustive search algorithm.

Organization of this paper. In section 2 we will move on to the basic ingredients that explain our techniques to solve the enumeration of IP problem. Then, in section 3, we give the two implications of enumerative problem. In section 4, we give the mathematic principle and the algorithms for counting the number of equivalence classes under IP with one secret problem. And In section 5, we present an example to illustrate our method and analyze the results.

2 Preliminaries

In this section, we remind the definition of the IP problem first given by J.Patarin [8] and the univariate representation of a polynomial system [9] which is crucial to our method. Then we recall the definition of friendly mapping introduced in [1] and some basic properties.

2.1 Isomorphism of Polynomials

By F_q we denote a finite field with q elements and by $F_q[\bar{x}] = F_q[x_1, \dots, x_n]$ the polynomial ring in the indeterminates $\bar{x} = x_1, \dots, x_n$ over F_q where $n > 1$. Let $u > 1$ be an integer and $A, B \in F_q[\bar{x}]^u$ such that all polynomials in $A = (a_1(\bar{x}), \dots, a_u(\bar{x}))$ and $B = (b_1(\bar{x}), \dots, b_u(\bar{x}))$ are of total degree 2. Then we say A and B are *isomorphic* if there are two invertible affine transformations $T = (T_L, T_C) \in GL_u(F_q) \times F_q^u, S = (S_L, S_C) \in GL_n F_q \times F_q^n$ satisfying $(b_1(\bar{x}), \dots, b_u(\bar{x})) = (a_1(\bar{x}S_L + S_C), \dots, a_u(\bar{x}S_L + S_C)) \cdot T_L + T_C$, i.e. $B = T \circ A \circ S$.

The IP problem can be stated as follows: given isomorphic $A, B \in F_q[\bar{x}]^u$ as above, find an isomorphism (T, S) from A to B. More precisely, this problem is also known as IP with two secrets (IP2S). There is another problem called IP with one secret (IP1S) in which we only consider the action of S and the degrees of polynomials in A and B may be greater than 2.

There are some variants according to the following parameters: the first one is that S and T are affine or linear; the second is that the polynomials in A and B are homogeneous or not; the third is that the number of indeterminates n equals to the number of polynomials u or not. These factors have influence on the difficulty of the IP problem to some degree. Note that the IP problem concerned in this paper is the linear, homogeneous and $n = u$ variant.

2.2 Univariate Representations

Isomorphism between F_{q^n} and F_q^n. Take $g(x) \in F_q[x]$ to be an irreducible polynomial of degree n, then $F_{q^n} \sim F_q[x]/g(x)$. It is well known that F_{q^n} as vector space over F_q and F_q^n are isomorphic. Let ϕ be the standard F_q- linear isomorphism between F_{q^n} and F_q^n given by

$$\phi(\alpha_0 + \alpha_1 x + \cdots + \alpha_{n-1}x^{n-1}) = (\alpha_1, \alpha_1, \cdots, \alpha_{n-1})$$

Using this map we can "lift" the quadratic polynomial system and linear transformation onto the extension field F_{q^n}.

Quadratic polynomial systems. We denote all systems of n homogeneous quadratic polynomials in n indeterminates over the ground field F_q by $\mathcal{P}$. Let $P \in \mathcal{P}$, then the univariate representation of P, $\bar{P} = \phi^{-1} \circ P \circ \phi$, is of the form:

$$\bar{P}(X) = \sum_{i=0}^{n-1} \sum_{j=0}^{i} \alpha_{ij} X^{q^i + q^j}$$

for some $\alpha_{ij} \in F_{q^n}$, if $q > 2$. We note that for $q = 2$ the correspondence dose not hold.

This correspond was first given by Kipnis and Shamir in [9]. Then we use $\bar{\mathcal{P}}$ to denote the corresponding univariate polynomials in $\mathcal{P}$.

Linear transformations. Let L be a linear transformation of F_q-vector space F_q^n, then $\bar{L} = \phi^{-1} \circ L \circ \phi$ is of the form:

$$\bar{L}(X) = \sum_{i=0}^{n-1} \alpha_i X^{q^i}$$

where $\alpha_i \in F_{q^n}$.

Then we denote the set of all invertible linear transformations over F_q by $\mathcal{L}$ and its corresponding univariate representation set by $\bar{\mathcal{L}}$. In the sequel, we consider the IP problem over the extension field F_{q^n} using their univariate representations.

2.3 Friendly Mapping

In [1], Lin et. al. introduced the definition of friendly mapping which is the bridge connecting the univariate polynomial over the extension field F_{q^n} and the matrix over F_{q^n} while converting the operation of composition of polynomials to the congruence transformation of matrices. Note that the composition of polynomials mentioned above means the composition of an univariate representation of a homogeneous quadratic polynomials system and an univariate representation of a linear transformation over the ground field F_q. Generally speaking, for the composition of any two univariate polynomials the friendly mapping does not have the property. And the definition of friendly mapping is given as follow:

Definition 1. *Let $\mathcal{M}_{n\times n}(F_{q^n})$ be the set of all $n \times n$ matrices over F_{q^n}. A mapping Ψ from $\bar{\mathcal{P}}$ to $\mathcal{M}_{n\times n}(F_{q^n})$ is called* **friendly mapping** *if for every $\bar{L} \in \bar{\mathcal{L}}$ and $\bar{P} \in \bar{\mathcal{P}}$, $\Psi(P \circ \bar{L}) = \hat{L}\Psi(P)\hat{L}'$, where "$'$" means the transpose of a matrix and $\hat{L}$ is a matrix associated with $\bar{L}$ over extension field F_{q^n} as follow*

$$\hat{L} = \begin{pmatrix} a_0 & a_{n-1}^q & \cdots & a_1^{q^{n-1}} \\ a_1 & a_0^q & \cdots & a_2^{q^{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1} & a_{n-2}^q & \cdots & a_0^{q^{n-1}} \end{pmatrix}_{n\times n}$$

The authors of [1] gave a candidate of friendly mapping. For any $\bar{P} = \sum_{i=0}^{n-1}\sum_{j=0}^{i} a_{ij}X^{q^i+q^j} \in \bar{\mathcal{P}}$, define $\Psi_1 : \bar{\mathcal{P}} \to \mathcal{M}_{n\times n}(F_{q^n})$ as

$$\Psi_1(\bar{P}) = \begin{pmatrix} 2a_{00} & a_{10} & \cdots & a_{n-1,0} \\ a_{10} & 2a_{11} & \cdots & a_{n-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & 2a_{n-1,n-1} \end{pmatrix}.$$

Sometimes, we also call $\Psi_1(\bar{P})$ the matrix associated with $\bar{P}$ or associated matrix to $\bar{P}$. And easily to check that for $\bar{P_1}, \bar{P_2} \in \mathcal{P}$, $\Psi_1(P_1 + P_2) = \Psi_1(P_1) + \Psi_1(P_2)$.

3 Enumerative problem of polynomial isomorphism

The enumerative problem of polynomial isomorphism is first brought in [1]. It actually consists of two different problems which have close connections to each other. One is counting the number of equivalent keys for a fixed central function, i.e. the number of solutions of IP problem. The other is counting the number of different schemes, i.e. the number of equivalence classes since isomorphism is an equivalence relation.

3.1 Enumeration of equivalent keys

We suppose that F_1 and F_2 are two central functions and isomorphic, then the IP problem for this instance is that find two invertible affine transformations (T, S) s.t. $F_2 = T \circ F_1 \circ S$. The goal is identify one pair(T, S), but the number of solutions of the instance is out of consideration. Here we will claim that the uniqueness of the solution is not always ture, that is there may exist more than one solution, say (T_1, S_1) and (T_2, S_2), i.e.

$$F_2 = T_1 \circ F_1 \circ S_1 = T_2 \circ F_1 \circ S_2$$

If we use F_1 as the central function of one scheme, then the effect of the two secret keys (T_1, S_1) and (T_2, S_2) are the same, that is they will generate the same public key, i.e. F_2. Hence we call (T_1, S_1) and (T_2, S_2) the **equivalent keys** for F_1. Therefore the number of solutions of IP problem is highly related to the real number of different secret keys for a fix central function.

From the equation above we have that

$$F_1 = (T_1^{-1} \circ T_2) \circ F_1 \circ (S_2 \circ S_1^{-1})$$

We use $Solution(F2, F1)$ to denote the solution of the instance of IP problem, i.e. $\{(T, S) \in (GL_u(F_q) \times F_q^u) \times (GL_n(F_q) \times F_q^n) | F_2 = T \circ F_1 \circ S\}$. If F_1 and F_2 are isomorphic, then it is easy to check that $|Solution(F_2, F_1)| = |Solution(F_1, F_1)|$. And $Solution(F_2, F_1) = \{(T \circ U, V \circ S) | (U, V) \in Solution(F_1, F_1)\}$ provided $(T, S) \in Solution(F_2, F_1)$.

The number of equivalent keys for isomorphic polynomials are the same, that is in one equivalence class, whichever polynomial you choose as the the central function, the number of equivalent keys is the same. And as above, for an equivalence class, we only need to identify the number of solutions of this instance of IP problem: $F_1 = T' \circ F_1 \circ S'$, i.e. $|Solution(F_1, F_1)|$.

3.2 Enumeration of equivalence classes

Since the isomorphism of polynomials is an equivalence relation, then this relation induces a partition of all homogeneous quadratic polynomials. The enumeration of equivalence classes is to identify the exact number of equivalence classes.

This number corresponds to the number of different schemes, since if we pick two polynomial systems from one equivalence class as the central functions, then we can always get the same public keys by delicate choices of the secret keys. That is if we pick two isomorphic systems of polynomials , say F_1 and F_2, as the central functions and $F_2 = T \circ F_1 \circ S$. If we arbitrarily choose (T', S') as the secret key for F_2, then we can choose $(T' \circ T, S \circ S')$ as the secret key for F_1, which makes the public keys equal, i.e.

$$T' \circ F_2 \circ S' = T' \circ (T \circ F_1 \circ S) \circ S' = (T' \circ T) \circ F_1 \circ (S \circ S')$$

Note that the formulation above is for IP2S problem, and there are the corresponding problems for IP1S problem. In the next section, we will give some techniques for identifying the exact number of equivalence classes for IP1S problem over even characteristic ground field F_q with exception $F_q = F_2$.

4 Decide the polynomial equivalence classes

In this section, we will introduce some techniques to identify all the polynomial equivalence classes. Let F_q be a finite field of characteristic 2, n be an integer and $\mathfrak{P}$ be the set of all homogeneous quadratic polynomial systems, i.e.

$$\mathfrak{P} = \left\{ \begin{pmatrix} p_1(x_1, \dots, x_n) \\ \cdots \\ p_n(x_1, \dots, x_n) \end{pmatrix} | \begin{matrix} p_i(x_1, \dots, x_n) \text{is a homogeneous} \\ \text{quadratic polynomial in} F_q[x_1, \dots, x_n] \end{matrix} \right\}$$

In order to classify the set $\mathfrak{P}$, we firstly "lift" all the polynomials systems and linear transformations over the ground field F_q onto the extension field F_{q^n}. By the standard isomorphism $\phi : F_{q^n} \to F_q^n$, we "lift" $\mathfrak{P}$ from F_q^n onto F_{q^n} which is $\phi^{-1} \circ \mathfrak{P} \circ \phi$, in which the polynomial has the form $\sum_{i=0}^{n-1} \sum_{j=0}^{i-1} a_{ij} X^{q^i + q^j} \in F_{q^n}[X]$, where $a_{ij} \in F_{q^n}$. In the sequel, we denote the set of polynomials over the extension field F_{q^n} by $\mathfrak{P}$ for simplicity. Note that here we deal with the case that $\mathrm{Ch}(F_q) = 2$ and $q > 2$.

Because we consider the problem of IP1S, for simplicity, as in [1], we have the following definition.

Definition 2. *Let $\bar{P}_1(X) = \sum_{i=0}^{n-1} \sum_{j=1}^{i-1} a_{ij} X^{q^i+q^j}$, $\bar{P}_2(X) = \sum_{i=0}^{n-1} \sum_{j=1}^{i-1} b_{ij} X^{q^i+q^j} \in \bar{\mathcal{P}}$. We say that $\bar{P}_1$ and $\bar{P}_2$ are* **linearly equivalent** *if and only if there exists $\bar{L}(X) = \sum_{i=0}^{n-1} a_i X^{q^i} \in \bar{\mathcal{L}}$ such that $\bar{P}_1(\bar{L}(X)) = \bar{P}_2(X)$ for all $X \in F_{q^n}$.*

4.1 An important observation

As we know the elements in $\mathfrak{P}$ is nothing but a linearly combination of monomials of the form $X^{q^i+q^j} \in F_{q^n}[X]$. By the definition of friendly mapping Ψ_1, we know that Ψ_1 maps monomials $X^{q^i+q^j}$ for $i \neq j$ and X^{2q^i} to triangular and diagonal entries of its associated matrix. Thus we can roughly classify the monomials $X^{q^i+q^j}$ into two sorts, one is $X^{q^i+q^j}$ for $i \neq j$ and the other is X^{2q^i}. According to corollary 4 of [1] monomials from the two sorts are not linearly isomorphic.

Therefore we suppose $\mathcal{T} = \{X^{q^i+q^j} | 0 \leq i < j \leq n-1\}$ and $\mathcal{D} = \{X^{2q^i} | 0 \leq i \leq n-1\}$. From the definition of friendly mapping it follows that Ψ_1 maps monomials in $\mathcal{T}$(resp. $\mathcal{D}$) to the triangular(resp. diagonal) part of the associated matrix, thus we call the monomials in $\mathcal{T}$(resp. $\mathcal{D}$) the triangular(resp. diagonal) monomial simply. And we use $n_{\mathcal{T}}$(resp. $n_{\mathcal{D}}$) to denote the cardinality of $\mathcal{T}$(resp. $\mathcal{D}$). It is obvious that $n_{\mathcal{T}} = \frac{1}{2}n(n-1)$ and $n_{\mathcal{D}} = n$.

We suppose

$$\mathcal{F} = \{\sum_{i=0}^{n-1}\sum_{j=0}^{i-1} a_{ij} X^{q^i+q^j} \in F_{q^n}[X] | a_{ij} \text{ can not be zero at the same time}\}$$

and

$$\mathcal{G} = \{\sum_{i=0}^{n-1} a_i X^{2q^i} \in F_{q^n}[X] | a_i \text{ can not be zero at the same time}\}$$

The set $\mathcal{F}$(resp. $\mathcal{G}$) is the linear combination of the monomials in $\mathcal{T}$(resp. $\mathcal{D}$) without the zero polynomial and we may call the polynomial in $\mathcal{F}$(resp. $\mathcal{G}$) triangular(resp. diagonal) polynomial simply. We use $n_{\mathcal{F}}$(resp. $n_{\mathcal{G}}$) to denote the cardinality of $\mathcal{F}$(resp. $\mathcal{G}$) and it is easy to compute that $n_{\mathcal{F}} = (q^n)^{\frac{1}{2}n(n-1)} - 1$ and $n_{\mathcal{G}} = (q^n)^n - 1$. And there is another class of polynomials which are the linear combination of monomial in $\mathcal{T}$ and $\mathcal{D}$, i.e.

$$\mathcal{M} = \{f + g | f \in \mathcal{F}, g \in \mathcal{G}\} = \mathcal{F} + \mathcal{G}$$

we may call the polynomial in $\mathcal{M}$ mixed polynomial. Similarly using $n_{\mathcal{M}}$ to denote the cardinality of $\mathcal{M}$, thus $n_{\mathcal{M}} = n_{\mathcal{F}} n_{\mathcal{G}}$. Therefor we can roughly classify $\mathfrak{P}$ into four disjoint sets, that is, $\{0\}, \mathcal{F}, \mathcal{G}$ and $\mathcal{M}$ where 0 is the zero polynomial, i.e.

$$\mathfrak{P} = \{0\} \cup \mathcal{F} \cup \mathcal{G} \cup \mathcal{M}$$

And easy to get that $n_{\mathfrak{P}} = 1 + n_{\mathcal{F}} + n_{\mathcal{G}} + n_{\mathcal{M}} = q^{\frac{1}{2}n^2(n+1)}$. Now we give the important observation.

Lemma 1. *For any polynomial $g \in \mathcal{G}$, to which the polynomials linearly equivalent are still in $\mathcal{G}$.*

Proof: suppose $g = \sum_{i=0}^{n-1} g_i X^{2q^i}$ and the invertible linear transformation $l \in \bar{\mathcal{L}}$ is $\sum_{i=0}^{n-1} a_i X^{q^i}$, thus

$$\begin{aligned} g \circ l &= (\sum_{i=0}^{n-1} g_i l^{2q^i}) \\ &= \sum_{i=0}^{n-1} g_i (\sum_{j=0}^{n-1} a_j X^{q^j})^{2q^i} \\ &= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} g_i a_j^{2q^i} X^{2q^{i+j}} \end{aligned}$$

Thus $g \circ l$ is a linear combination of monomials in $\mathcal{D}$, that is, $g \circ l \in \mathcal{G}$. □

This lemma implies that we can consider the classification of $\mathcal{G}$ separately.

Theorem 1. *Let h_1 and h_2 be two polynomials in $\mathfrak{P}$. If they are linearly isomorphic then $\Psi_1(h_1)$ and $\Psi_1(h_2)$ are congruent.*

Proof: h_1 and h_2 are linearly isomorphic if and only if there exists a linear transformation $l \in \bar{\mathcal{L}}$, s.t. $h_1 \circ l = h_2$. By the definition of Ψ_1 it follows that $\Psi_1(h_2) = \Psi_1(h_1 \circ l) = \hat{l}\Psi_1(h_1)\hat{l}'$, thus $\Psi_1(h_1)$ and $\Psi(h_2)$ are congruent. Here $\hat{l}$ is the matrix corresponding to linear transformation l. □

Remark 1: The inverse of the theorem is not right in general;

Remark 2: If we divide h_i into the triangular part and diagonal part, say $h_i = f_i + g_i$ where $f_i \in \mathcal{F} \cup \{0\}$ and $g_i \in \mathcal{G} \cup \{0\}$ for $i = 1, 2$, as $\Psi_1(h_i) = \Psi_1(f_i + g_i) = \Psi_1(f_i) + \Psi_1(g_i) = \Psi_1(f_i)$ for $i = 1, 2$, then that $\Psi_1(h_1)$ and $\Psi(h_2)$ are congruent is equivalent to that $\Psi_1(f_1)$ and $\Psi(f_2)$ are congruent. This leads to the following concept.

Definition 3. *For any $h_1, h_2 \in \mathfrak{P}$, h_1 and h_2 are called matrix isomorphic if $\Psi_1(h_1)$ and $\Psi_1(h_2)$ are congruent.*

By the definition 3 and theorem 1 we can say that matrix isomorphism is necessary condition of linearly isomorphism.

Lemma 2. *For any $h_1, h_2 \in \mathfrak{P}$, $\Psi_1(h_1)$ and $\Psi_1(h_2)$ are congruent if and only if there exists an invertible linear transformation $l \in \bar{\mathcal{L}}$ s.t. $h_1 \circ l = h_2 + g$ for some $g \in \mathcal{G} \cup \{0\}$.*

Proof: (sufficiency) $\Psi_1(h_2) = \Psi_1(h_2) + \Psi_1(g) = \Psi_1(h_2 + g) = \Psi_1(h_1 \circ l) = \hat{l}\Psi_1(h_1)\hat{l}'$, thus $\Psi_1(h_1)$ and $\Psi_1(h_2)$ are congruent.

(necessity) Because $\Psi_1(h_1)$ and $\Psi_1(h_2)$ are congruent, there exists an invertible linear transformation, say $l \in \bar{\mathcal{L}}$, s.t. $\hat{l}\Psi_1(h_1)\hat{l}' = \Psi_1(h_2)$. Thus $\Psi_1(h_1 \circ l) = \Psi_1(h_2)$, i.e. $\Psi_1(h_1 \circ l) + \Psi_1(h_2) = \Psi_1(h_1 \circ l + h_2) = O$, where O is the zero matrix. Therefore $h_1 \circ l + h_2$ must be in $\mathcal{G} \cup \{0\}$ by the definition of Ψ_1, thus we suppose $g = h_1 \circ l + h_2$, i.e. $h_1 \circ l = h_2 + g$ for some $g \in \mathcal{G} \cup \{0\}$. Then the lemma is shown. □

Lemma 3. *Matrix isomorphism defined in definition 3 is an equivalence relation.*

Proof:

1. (reflexivity) for any polynomial $h \in \mathfrak{P}$, h and h are matrix isomorphic by X;
2. (symmetry) for $h_1, h_2 \in \mathfrak{P}$ and h_1 is matrix isomorphic to h_2, i.e. there exists an invertible linear transformation $l \in \mathcal{L}$ s.t. $h_1 \circ l = h_2 + g$ for some $g \in \mathcal{G} \cup \{0\}$. By invertibility of l it follows that $h_2 \circ l^{-1} = (h_1 \circ l + g) \circ l^{-1} = h_1 \circ l \circ l^{-1} + g \circ l^{-1} = h_1 + g \circ l^{-1}$. By lemma 1 $g \circ l^{-1}$ is still in $\mathcal{G} \cup \{0\}$. Thus h_2 is also matrix isomorphic to h_2;
3. (transitivity) for $h_1, h_2, h_3 \in \mathfrak{P}$, if h_1 is matrix isomorphic to h_2 and h_2 is matrix isomorphic to h_3, i.e. there exist two invertible linear transformations $l_1, l_2 \in \mathcal{L}$ s.t. $h_1 \circ l_1 = h_2 + g_1$ and $h_2 \circ l_2 = h_3 + g_2$ for some $g_1, g_2 \in \mathcal{G} \cup \{0\}$. Then $h_1 \circ l_1 \circ l_2 = (h_2 + g_1) \circ l_2 = h_2 \circ l_2 + g_1 \circ l_2 = h_3 + g_2 + g_1 \circ l_2$, it is obvious that $g_2 + g_1 \circ l_2$ is in $\mathcal{G} \cup \{0\}$. □

From lemma 3 we know that matrix isomorphic is also an equivalence relation and a necessary condition of linearly isomorphism. Thus it is straightforward to have the idea that we can classify $\mathfrak{P}$ using the matrix isomorphism, then in each equivalence class using linearly isomorphism we can make a finer classification.

4.2 The main technique

In this subsection we give the principle to classify each matrix equivalence class using the linearly isomorphism equivalence relation.

By the property of friendly mapping which is shown in the remark 2 of theorem 1 it follows that $\Psi_1(\mathfrak{P}/(\mathcal{G} \cup \{0\})) = \Psi_1(\mathcal{F})$, thus using matrix equivalence relation that we classify $\mathfrak{P}/(\mathcal{G} \cup \{0\})$ is equivalent to classify $\mathcal{F}$. We suppose that $\mathcal{F} = \bigcup_{i=1}^{m_{\mathcal{F}}} \mathcal{F}_i$ and $\mathcal{F}_i \cap \mathcal{F}_j = \phi$ for $i \neq j$. Thus $\{\mathcal{F}_1, \ldots, \mathcal{F}_{m_{\mathcal{F}}}\}$ is a partition of $\mathcal{F}$, i.e. a partition of $\mathfrak{P}/(\mathcal{G} \cup \{0\})$, where $m_{\mathcal{F}}$ is the number of matrix equivalence classes. $\mathfrak{P}/(\mathcal{G} \cup \{0\}) = \bigcup_{i=1}^{m_{\mathcal{F}}} (\mathcal{F}_i + \mathcal{G} \cup \{0\})$ where $\mathcal{F}_i + \mathcal{G} \cup \{0\} = \{f + g | f \in \mathcal{F}_i, g \in \mathcal{G} \cup \{0\}\}$.

However how to get the partition is not the focal point of this paper, we focus on that based on this rough classification of $\mathfrak{P}$ how to get the finer partition under linearly isomorphic equivalence relation, that is how to classify the set $\mathcal{F}_i + \mathcal{G} \cup \{0\}$. Next we will introduce the main technique.

Theorem 2. *If $f_1 \in \mathcal{F}_i$ and $f_2 \in \mathcal{F}_j$ with $i \neq j$, then for any $g_1, g_2 \in \mathcal{G} \cup \{0\}$, $f_1 + g_1$ can not be linearly isomorphic to $f_2 + g_2$.*

Proof: For contradiction we suppose that $f_1 + g_1$ is linearly isomorphic to $f_2 + g_2$ by invertible linear transformation $l \in \bar{\mathcal{L}}$, i.e. $(f_1 + g_1) \circ l = f_2 + g_2$, thus $f_1 \circ l = f_2 + (g_1 \circ l + g_2)$. From lemma 1 it follows that $(g_1 \circ l + g_2)$ is still in $\mathcal{G} \cup \{0\}$, thus f_1 and f_2 are matrix isomorphic which means they are in the same matrix equivalence class. That is contradiction to the condition that $f_1 \in \mathcal{F}_i$ and $f_2 \in \mathcal{F}_j$ with $i \neq j$. The theorem is shown. □

Remark: The theorem indicates that polynomials from different matrix equivalence classes can not be linearly isomorphic.

Theorem 3. *If f_1 and f_2 are in the same matrix equivalence class, say $\mathcal{F}_i$, then for any $g \in \mathcal{G} \cup \{0\}$, the linearly equivalence class of $f_2 + g$, i.e. the set $\{(f_2 + g) \circ l | l \in \bar{\mathcal{L}}\}$, must contain $f_1 + g'$ for some $g' \in \mathcal{G} \cup \{0\}$.*

Proof: As f_1 and f_2 are matrix isomorphic, there exists a invertible linear transformation $l \in \bar{\mathcal{L}}$ s.t. $f_2 \circ l = f_1 + g_1$ for some $g_1 \in \mathcal{G} \cup \{0\}$. Thus $(f_2 + g) \circ l = f_2 \circ l + g \circ l = f_1 + (g_1 + g \circ l)$. By lemma 1 $(g_1 + g \circ l)$ is in $\mathcal{G} \cup \{0\}$, we denote $(g_1 + g \circ l)$ by g'. Therefore $(f_2 + g) \circ l = f_1 + g'$ for some $g' \in \mathcal{G} \cup \{0\}$. □

The theorem shown above indicates that if we choose an arbitrary element $f_i \in \mathcal{F}_i$ as the representative of $\mathcal{F}_i$, then every linearly equivalence class of any element in $\mathcal{F}_i$ contains a polynomial of the form $f_i + g$ for some $g \in \mathcal{G} \cup \{0\}$. We can consider $f_i + g$ as the representative of its linearly equivalence class. Thus

$$\mathcal{F}_i + \mathcal{G} \cup \{0\} = \bigcup_{j=1}^{m_{\mathcal{F}_i}} \overline{f_i + g_j}$$

where $\overline{f_i + g_j}$ is the linearly equivalence class containing $f_i + g_j$, i.e. $f_i + g_j = \{(f_i + g_j) \circ l | l \in \bar{\mathcal{L}}\}$and $m_{\mathcal{F}_i}$ is the number of linearly equivalence classes in $\mathcal{F}_i + \mathcal{G} \cup \{0\}$. If we want to determine $m_{\mathcal{F}_i}$, the number of linearly equivalence classes in $\mathcal{F}_i + \mathcal{G} \cup \{0\}$, we have to identify the exact number of all linearly equivalence classes of $f_i + g$ for $\underline{\text{any } g \in \mathcal{G} \cup \{0\},}$ that is we have to determine for different $g_1, g_2 \in \mathcal{G} \cup \{0\}$, if $\overline{f_i + g_1}$ and $\overline{f_i + g_2}$ represent the same class.

Therefore we don't directly compute the number of linearly equivalence classes in each matrix equivalence class, instead we try to count how many classes the diagonal polynomials are classified into. The latter problem is much easier.

For simplicity we use the notation $\mathrm{Stab}(\Psi_1(f_i))$ to denote the set of invertible linear transformations $l \in \bar{\mathcal{L}}$ s.t. $\hat{l}\Psi_1(f_i)\hat{l}' = \Psi_1(f_i)$, i.e.

$$\begin{aligned}\mathrm{Stab}(\Psi_1(f_i)) &= \{l \in \bar{\mathcal{L}} | \hat{l}\Psi_1(f_i)\hat{l}' = \Psi_1(f_i)\} \\ &= \{l \in \bar{\mathcal{L}} | \Psi_1(f_i \circ l) = \Psi_1(f_i)\} \\ &= \{l \in \bar{\mathcal{L}} | f_i \circ l = f_i + g \text{ for some } g \in \mathcal{G} \cup \{0\}\}\end{aligned}$$

and use the similar notation $\mathrm{Stab}(f_i)$ to denote $\{l \in \bar{\mathcal{L}} | f_i \circ l = f_i\}$.

Theorem 4. *If $h \in \mathfrak{P}$, then*

(i) $\mathrm{Stab}(\Psi_1(h)) \leq \bar{\mathcal{L}}$;
(ii) $\mathrm{Stab}(h) \leq \bar{\mathcal{L}}$;
(iii) $\mathrm{Stab}(h) \leq \mathrm{Stab}(\Psi_1(h))$.

Proof: (i) We suppose $l_1, l_2 \in \mathrm{Stab}(\Psi_1(h))$, i.e. $h \circ l_1 = h + g_1$ and $h \circ l_2 = h + g_2$ for some $g_1, g_2 \in \mathcal{G} \cup \{0\}$. Then $h \circ (l_1 \circ l_2^{-1}) = (h \circ l_1) \circ l_2^{-1} = (h + g_1) \circ l_2^{-1} = h \circ l_2^{-1} + g_1 \circ l_2^{-1} = h + g_2 \circ l_2^{-1} + g_1 \circ l_2^{-1} = h + (g_1 + g_2) \circ l_2^{-1}$. By lemma 1 $(g_1 + g_2) \circ l_2^{-1}$ is in $\mathcal{G} \cup \{0\}$, thus $l_1 \circ l_2^{-1}$ is also in $\mathrm{Stab}(\Psi_1(h))$ which means $\mathrm{Stab}(\Psi_1(h))$ forms a group with respect to the usual composition of mappings.

(ii) The proof is immediate.
(iii) By (i) and (ii) it is sufficient to show that $\mathrm{Stab}(h) \subseteq \mathrm{Stab}(\Psi_1(h))$ which is obvious. $\square$

Thus to achieve the goal of determining the exact number of linear equivalence classes of $\mathcal{F}_i + \mathcal{G} \cup \{0\}$, for a given $g \in \mathcal{G} \cup \{0\}$ we have to determine how many $g' \in \mathcal{G} \cup \{0\}$ s.t. $f_i + g'$ are in the linearly equivalence class containing $f_i + g$, that is equivalent to determine how many $\overline{f_i + g}$ represent the same linearly equivalence class.

Corollary 1. *If f_i is a representative of matrix equivalence class $\mathcal{F}_i$ and $g \in \mathcal{G} \cup \{0\}$, then* $\mathrm{Stab}(f_i + g) \leq \mathrm{Stab}(\Psi_1(f_i))$.

Proof: By theorem 4 (i) and (ii), $\mathrm{Stab}(\Psi_1(f_i)) \leq \bar{\mathcal{L}}$ and $\mathrm{Stab}(f_i + g) \leq \bar{\mathcal{L}}$. Thus it is sufficient to show that $\mathrm{Stab}(f_i + g) \subseteq \mathrm{Stab}(\Psi_1(f_i))$. For any element $l \in \mathrm{Stab}(f_i + g)$, we have that $(f_i + g) \circ l = f_i + g$. Thus $f_i \circ l = f_i + (g + g \circ l)$. By lemma 1 $g + g \circ l$ is in $\mathcal{G} \cup \{0\}$ which means l is in $\mathrm{Stab}(\Psi_1(f_i))$. Then the corollary is shown. $\square$

Theorem 5. *If f_i is a representative of matrix equivalence class $\mathcal{F}_i$, then for a given $g' \in \mathcal{G} \cup \{0\}$, the cardinality of $\{g \in \mathcal{G} \cup \{0\} | \overline{f_i + g} = \overline{f_i + g'}\}$ is* $[\mathrm{Stab}(\Psi_1(f_i)) : \mathrm{Stab}(f_i + g')] = |\mathrm{Stab}(\Psi_1(f_i))| / |\mathrm{Stab}(f_i + g')|$.

Proof: By corollary 1 $\mathrm{Stab}(f_i + g') \leq \mathrm{Stab}(\Psi_1(f_i))$. We suppose that l_1, l_2 are in the same right coset of $\mathrm{Stab}(\Psi_1(f_i))$ modulo $\mathrm{Stab}(f_i + g')$, that is, there exist a linear transformation $l \in \mathrm{Stab}(f_i + g')$ s.t. $l_1 = l \circ l_2$. Then $(f_i + g') \circ l_1 = (f_i + g') \circ (l \circ l_2) = (f_i + g') \circ l_2$. And the triangular parts of $(f_i + g') \circ l_1$ and $(f_i + g') \circ l_2$ are the same, namely f_i. Thus the diagonal parts must equal, which means linear transformations in the same coset will "generate" the same diagonal polynomial. Then by Lagrange Theorem we can get the conclusion. $\square$

4.3 The algorithms

In this subsection we give two algorithms to identify the number of matrix equivalence classes and linearly equivalence classes of $\mathfrak{P}$ according to the technique given in the previous subsection.

Algorithm 1: Identify all matrix equivalence classes

```
Input: $\mathcal{F}$
Output: $R, C = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_{m_\mathcal{F}}\}$ and $S$
1: $R \leftarrow \phi, C \leftarrow \phi, S \leftarrow \phi, B \leftarrow \mathcal{F}$
2: while not $IsEmpty(B)$ do
3:    f = Random($B$)
4:    $R \leftarrow R \cup \{f\}; \mathcal{F}_{temp} \leftarrow \{f\}; S_{temp} \leftarrow \phi$
5:    for $l \in \mathcal{L}$ do
6:       $h' = f \circ l$
7:       if $TriangularPart(h') = f$ then
```

```
8:          $S_{temp} \leftarrow S_{temp} \cup \{l\}$
9:        else  $\mathcal{F}_{temp} \leftarrow \mathcal{F}_{temp} \cup \{TriangularPart(h')\}$
10:       end if
11:    end for
12:    $C \leftarrow C \cup \{\mathcal{F}_{temp}\}; S \leftarrow S \cup \{S_{temp}\}$
13:    $B \leftarrow B \backslash \mathcal{F}_{temp}$
14: end while
15: return $(R, C, S)$
```

In algorithm 1, $\mathcal{F}$ is the set of triangular polynomials as given in subsection 4.1. R contains the representatives of $\mathcal{F}_i (1 \leq i \leq m_{\mathcal{F}})$, i.e. f_i, C contains all matrix equivalence classes and S contains the stabilizer of $\Psi_1(f_i)$. It is easy to see that algorithm 1 is actually exhaustive search in $\mathcal{F}$. Obviously, in this way, we not only get the number of matrix equivalence classes, but also get exact every matrix equivalence class i.e. C and the matrix stabilizer for every representative i.e. S.

By corollary 1, we know that if $|\mathrm{Stab}(\Psi_1(f_i))| = 1$, then for any $g \in \mathcal{G} \cup \{0\}$, $|\mathrm{Stab}(\Psi_1(f_i + g))| = 1$. Thus we can omit the process for searching the stabilizer of $f_i + g$. For the f_i with $|\mathrm{Stab}(\Psi_1(f_i))| > 1$, we have the following algorithm to identify the number of linearly equivalence classes in $\mathcal{F}_i + \mathcal{G} \cup \{0\}$.

Algorithm 2: Classify each matrix equivalence class using our techniques

```
Input: $f_i, S_i, \mathcal{G} \cup \{0\}$
Output: $R, C = \{G_1, G_2, \ldots, G_{m_{\mathcal{F}_i}}\}$ and $S$
1: $R \leftarrow \phi, C \leftarrow \phi, S \leftarrow \phi, B \leftarrow \mathcal{G} \cup \{0\}$
2: while not $IsEmpty(B)$ do
3:    g = Random($B$)
4:    $R \leftarrow R \cup \{g\}; G_{temp} \leftarrow \{g\}; S_{temp} \leftarrow \phi$
5:    for $l \in S_i$ do
6:       $h' = (f_i + g) \circ l$
7:       if $h' = f_i + g$ then
8:          $S_{temp} \leftarrow S_{temp} \cup \{l\}$
9:       else  $G_{temp} \leftarrow G_{temp} \cup \{h' - f_i\}$
10:      end if
11:   end for
12:   $C \leftarrow C \cup \{G_{temp}\}; S \leftarrow S \cup \{S_{temp}\}$
13:   $B \leftarrow B \backslash G_{temp}$
14: end while
15: return $(R, C, S)$
```

In algorithm 2, based on the output of algorithm 1 we identify the number of linearly equivalence classes in each matrix equivalence class. The notions are as in algorithm 1. And since we have identified the set $\mathrm{Stab}(\varPsi_1(f_i))$, when we try to identify the stabilizer of $f_i + g$ for some $g \in \mathcal{G} \cup \{0\}$ we can only check the linear transformations in $\mathrm{Stab}(\varPsi_1(f_i))$. Empirically $|\mathrm{Stab}(\varPsi_1(f_i))|$ is far less than $|\bar{\mathcal{L}}|$, thus the efficiency is higher than the exhaustive search algorithm.

Argument for the efficiency
From the proof of lemma 7 in [1] we get this theorem

Theorem 6. *For any $a \in F_{q^n}^*$ and $0 \le j < i \le n-1$,*

$$|Stab(\varPsi_1(aX^{q^i+q^j}))| = |Stab(\varPsi_1(X^{q^{i-j}+1}))| = |\{cX|c^{q^{i-j}+1} = 1\}|$$

except $i - j = \frac{n}{2}$.

Proof: The theorem follow easily from that for $i - j = \frac{n}{2}$,

$$\mathrm{Stab}(\varPsi_1(aX^{q^i+q^j})) = \mathrm{Stab}(\varPsi_1(X^{q^i+q^j}))$$

and

$$\mathrm{Stab}(\varPsi_1(X^{q^i+q^j})) = X^{q^{-j}} \circ \mathrm{Stab}(\varPsi_1(X^{q^{i-j}+1})) \circ X^{q^i}$$

□

And we know that for $X^{q^k+1} = 1$ has only one solution in F_{q^n} if and only if $\frac{n}{\gcd(k,n)}$ is odd. Thus we suppose $n = 2^s n_0$ and n_0 is not divisible by 2. Then for those $k = 2^s l$ where $1 \le l \le n_0 - 1$, $X^{q^k+1} = 1$ has only one solution, i.e. $X = 1$.

Therefore by theorem 6, for monomials of the form $aX^{q^i+q^j}$ where $a \in F_{q^n}^*$ and $0 \le j < i \le n-1$ s.t. $\frac{n}{\gcd(i-j,n)} = 1$, the order of matrix stabilizer is 1. By remark 2 of theorem 1 we know that for any polynomial $h = f + g$ where f is a monomial discussed above and $g \in \mathcal{G}\{0\}$, $|\mathrm{Stab}(\varPsi_1(h))| = 1$. And the number of these kinds of polynomials is $\frac{1}{2}(n_0 - 1)n(q^n - 1)q^{n^2}$, which means using the algorithms given in this paper we need not check at least $\frac{1}{2}(n_0 - 1)n(q^n - 1)q^{n^2}$ polynomials.

5 Experiments and Analysis

In this Section, we will give an example to show how to use our technique to find all the polynomial isomorphism classes. In this example, we will restrict our polynomial to be over finite field F_4.

Let $\mathfrak{P}$ be the set of all homogeneous quadratic polynomial systems over F_4 with 3 variables and 3 polynomials, that is

$$\mathfrak{P} = \left\{ \begin{bmatrix} f_1(x_1, x_2, x_3) \\ f_2(x_1, x_2, x_3) \\ f_3(x_1, x_2, x_3) \end{bmatrix} \middle| \begin{array}{c} f_i(x_1, x_2, x_3) \in F_4[x_1, x_2, x_3] \\ \text{is homogeneous and quadratic} \end{array} \right\}$$

Following our convention, we "lift" $\mathfrak{P}$ from F_4^3 onto extension field F_{4^3} by the standard isomorphism $\phi : F_{4^3} \to F_4^3$. In $\phi^{-1} \circ \mathfrak{P} \circ \phi$, the univariate polynomial has the form $\sum_{i=0}^{2}\sum_{j=0}^{i} a_{ij}X^{4^i+4^j}$. Then we use the notation $\mathfrak{P}$ to denote the set $\{\sum_{i=0}^{2}\sum_{j=0}^{i} a_{ij}X^{4^i+4^j} \in F_{4^3}[X]\}$.

Using the technique introduced in last section, we can divide $\mathfrak{P}$ into four parts which are

$$\begin{aligned}\mathcal{F} &= \{aX^{4^0+4^1} + bX^{4^0+4^2} + cX^{4^1+4^2} \in F_{4^3}[X] | a, b, c \text{ can not be zero at the same time}\}\\ &= \{aX^5 + bX^{17} + cX^{20} \in F_{4^3}[X] | a, b, c \text{ can not be zero at the same time}\}\end{aligned}$$

$$\begin{aligned}\mathcal{G} &= \{aX^{2\cdot 4^0} + bX^{2\cdot 4^1} + cX^{2\cdot 4^2} \in F_{4^3}[X] | a, b, c \text{ can not be zero at the same time}\}\\ &= \{aX^2 + bX^8 + cX^{32} \in F_{4^3}[X] | a, b, c \text{ can not be zero at the same time}\}\end{aligned}$$

$\mathcal{M} = \{f + g | f \in \mathcal{F}, g \in \mathcal{G}\} = \mathcal{F} + \mathcal{G}$ and $\{0\}$.

It is obvious that $n_{\mathcal{F}} = n_{\mathcal{G}} = (4^3)^3 - 1$ and $n_{\mathcal{M}} = (4^9 - 1)(4^9 - 1)$. Thus the cardinality of $\mathfrak{P}$ is $n_{\mathcal{F}} + n_{\mathcal{F}} + n_{\mathcal{F}} + 1 = 4^{18} = 2^{36}$. What we will do is to determine the number of linearly equivalence classes of the 2^{36} polynomials in $\mathfrak{P}$. And it is known that $|GL_3(F_4)| = 181440$.

5.1 Experiment results

Now we show the results of the example:

Step 1 Classify $\mathfrak{P}$ using matrix equivalence relation.
As we know polynomials in $\mathcal{G} \cup \{0\}$ can not be linearly equivalent to polynomials in $\mathcal{F}$ and $\mathcal{M}$. And the polynomials in $\mathcal{G} \cup \{0\}$ are all in one matrix equivalence class. Therefore in the first step actually we classify $\mathfrak{P}/(\mathcal{G} \cup \{0\})$, i.e. $\mathcal{F} \cup \mathcal{M}$, using matrix equivalence relation. For $\Psi_1(\mathcal{F}) = \Psi_1(\mathcal{F})$, actually we classify $\mathcal{F}$.

Experiment results show that $\mathcal{F}$ is divided into 43 matrix equivalence classes, suppose that

$$\mathcal{F} = \mathcal{A}_1 \bigcup_{i=1}^{21} \mathcal{B}_i \bigcup_{j=1}^{21} \mathcal{C}_j$$

where $|\text{Stab}(\Psi_1(\mathcal{A}_1))| = 1, |\text{Stab}(\Psi_1(\mathcal{B}_i))| = 48$ and $|\text{Stab}(\Psi_1(\mathcal{C}_j))| = 2880$ for $1 \le i, j \le 21$. Note that the notation $\text{Stab}(\Psi_1(\mathcal{A}_1))$ denote $\text{Stab}(\Psi_1(h))$ for any polynomial $h \in \mathcal{A}_1$ and it is the same for $\text{Stab}(\Psi_1(\mathcal{B}_i))$ and $\text{Stab}(\Psi_1(\mathcal{C}_j))$.

By theorem 4 (i) we know that

$$\begin{aligned}|\mathcal{A}_1| &= |GL_3(F_4)|/|\text{Stab}(\Psi_1(\mathcal{A}_1))| = 181440/1 = 181440\\ |\mathcal{B}_i| &= |GL_3(F_4)|/|\text{Stab}(\Psi_1(\mathcal{B}_i))| = 181440/48 = 3780\\ |\mathcal{C}_j| &= |GL_3(F_4)|/|\text{Stab}(\Psi_1(\mathcal{C}_j))| = 181440/2880 = 63\end{aligned}$$

for $1 \le i, j \le 21$. And we can compute that

$$\begin{aligned}&|\mathcal{A}_1| + \sum_{i=0}^{21}|\mathcal{B}_i| + \sum_{j=0}^{21}|\mathcal{C}_j|\\ &= 1 \times 181440 + 21 \times 3780 + 21 \times 63 = 262143 = 4^9 - 1\\ &= |\mathcal{F}|\end{aligned}$$

Step 2 Classify $\{\mathcal{A}_1 + \mathcal{G} \cup \{0\}\}$ using linearly equivalence relation.
By the analysis in previous section and $|\mathrm{Stab}(\Psi_1(\mathcal{A}_1))| = 1$ we know that for any polynomial in $\{\mathcal{A}_1 + \mathcal{G} \cup \{0\}\}$, say $h = f + g$ with $f \in \mathcal{A}_1$ and $g \in \{\mathcal{G} \cup \{0\}\}$, $|\mathrm{Stab}(h)| = 1$. Thus every element in $\{\mathcal{A}_1 + \mathcal{G} \cup \{0\}\}$ is in different linearly equivalence class which has exact 181440 polynomials, which means $\{\mathcal{A}_1 + \mathcal{G} \cup \{0\}\}$ is divided into 4^9 linearly equivalence classes.

Step 3 Classify $\{\mathcal{B}_i + \mathcal{G} \cup \{0\}\}$ for $1 \le i \le 21$ using linearly equivalence relation.
It is easy to compute that the cardinality of $\{\mathcal{B}_i + \mathcal{G} \cup \{0\}\}$ is $|\mathcal{B}_i| \times |\mathcal{G} \cup \{0\}| = 3780 \times 4^9$ for $1 \le i \le 21$.

Experiment results show that $\{\mathcal{B}_i + \mathcal{G} \cup \{0\}\}$ is divided into 6742 linearly equivalence classes, which are in four sorts. There are 160 linearly equivalence classes in the first sort and the order of polynomial stabilizer of the elements in any of these classes is 4. In the second sort, there is only one linearly equivalence class whose polynomial stabilizer has 3 elements. In the third sort, there are 2320 linearly equivalence classes and any class has polynomial stabilizer with order 2. At last in the forth sort, there are 4261 linearly equivalence classes and the polynomial stabilizer of each class has only one element, i.e. the identical transformation X. Thus we can suppose that

$$\{\mathcal{B}_i + \mathcal{G} \cup \{0\}\} = \bigcup_{j_1=1}^{160} B_{i,j_1}^{(1)} \bigcup B_{i,1}^{(2)} \bigcup_{j_3=1}^{2320} B_{i,j_3}^{(3)} \bigcup_{j_4=1}^{4261} B_{i,j_4}^{(4)}$$

for $1 \le i \le 21$. We easily get that

1. $|B_{i,j_1}^{(1)}| = 181440/4$ for $1 \le j_1 \le 160$;
2. $|B_{i,1}^{(2)}| = 181440/3$;
3. $|B_{i,j_3}^{(3)}| = 181440/2$ for $1 \le j_3 \le 2320$;
4. $|B_{i,j_4}^{(4)}| = 181440/1$ for $1 \le j_4 \le 4261$;

Thus the number of polynomials in the 6742 linearly equivalence classes is

$$160 \times \frac{181440}{4} + 1 \times \frac{181440}{3} + 2320 \times \frac{181440}{2} + 4261 \times 181440 = 3780 \times 4^9$$

which means

$$|\{\mathcal{B}_i + \mathcal{G} \cup \{0\}\}| = \sum_{j_1=1}^{160} |B_{i,j_1}^{(1)}| + |B_{i,1}^{(2)}| + \sum_{j_3=1}^{2320} |B_{i,j_3}^{(3)}| + \sum_{j_4=1}^{4261} |B_{i,j_4}^{(4)}|$$

Step 4 Classify $\{\mathcal{C}_j + \mathcal{G} \cup \{0\}\}$ for $1 \le j \le 21$ using linearly equivalence relation.
It is easy to compute that the cardinality of $\{\mathcal{C}_j + \mathcal{G} \cup \{0\}\}$ is $|\mathcal{C}_j| \times |\mathcal{G} \cup \{0\}| = 63 \times 4^9$ for $1 \le j \le 21$.

Experiment results show that $\{\mathcal{C}_j + \mathcal{G} \cup \{0\}\}$ is divided into 274 linearly equivalence classes, which are in ten sorts. In the equation below we use the notation $C^{(k)}$ to denote the equivalence classes in the kth sort:

$$\{\mathcal{C}_j + \mathcal{G} \cup \{0\}\} = \bigcup C^{(1)}_{j,1} \bigcup C^{(2)}_{j,1} \bigcup_{i_3=1}^{30} C^{(3)}_{j,i_3} \bigcup C^{(4)}_{j,1} \bigcup_{i_5=1}^{53} C^{(5)}_{j,i_5}$$
$$\bigcup_{i_6=1}^{20} C^{(6)}_{j,i_6} \bigcup_{i_7=1}^{20} C^{(7)}_{j,i_7} \bigcup_{i_8=1}^{5} C^{(8)}_{j,i_8} \bigcup_{i_9=1}^{120} C^{(9)}_{j,i_9} \bigcup_{i_{10}=1}^{23} C^{(10)}_{j,i_{10}}$$

and

1. $|C^{(1)}_{j,1}| = 181440/480$;
2. $|C^{(2)}_{j,1}| = 181440/288$;
3. $|C^{(3)}_{j,i_3}| = 181440/96$ for $1 \le i_3 \le 30$;
4. $|C^{(4)}_{j,1}| = 181440/60$;
5. $|C^{(5)}_{j,i_5}| = 181440/48$ for $1 \le i_5 \le 53$;
6. $|C^{(6)}_{j,i_6}| = 181440/10$ for $1 \le i_6 \le 20$;
7. $|C^{(7)}_{j,i_7}| = 181440/6$ for $1 \le i_7 \le 20$;
8. $|C^{(8)}_{j,i_8}| = 181440/4$ for $1 \le i_8 \le 5$;
9. $|C^{(9)}_{j,i_9}| = 181440/2$ for $1 \le i_9 \le 120$;
10. $|C^{(10)}_{j,i_{10}}| = 181440$ for $1 \le i_{10} \le 23$;

Thus the number of polynomials in the 274 linearly equivalence classes is

$$181440 \times (\frac{1}{480} + \frac{1}{288} + \frac{30}{96} + \frac{1}{60} + \frac{53}{48} + \frac{20}{10} + \frac{20}{6} + \frac{5}{4} + \frac{120}{2} + \frac{23}{1}) = 63 \times 4^9$$

which means

$$|\{\mathcal{C}_j + \mathcal{G} \cup \{0\}\}| = |C^{(1)}_{j,1}| + |C^{(2)}_{j,1}| + \sum_{i_3=1}^{30} |C^{(3)}_{j,i_3}| + |C^{(4)}_{j,1}| + \sum_{i_5=1}^{53} |C^{(5)}_{j,i_5}| + \sum_{i_6=1}^{20} |C^{(6)}_{j,i_6}|$$
$$+ \sum_{i_7=1}^{20} |C^{(7)}_{j,i_7}| + \sum_{i_8=1}^{5} |C^{(8)}_{j,i_8}| + \sum_{i_9=1}^{120} |C^{(9)}_{j,i_9}| + \sum_{i_{10}=1}^{23} |C^{(10)}_{j,i_{10}}|$$

Step 5 Classify $\mathcal{G} \cup \{0\}$ using linearly equivalence relation.
Experiments show that $\mathcal{G} \cup \{0\}$ is divided into 44 linearly equivalence classes which are in four sorts:

$$\mathcal{G} \cup \{0\} = G^{(1)}_1 \bigcup_{i=1}^{21} G^{(2)}_i \bigcup_{j=1}^{21} G^{(3)}_j \bigcup \{0\}$$

and

1. $|G^{(1)}_1| = 181440$;
2. $|G^{(2)}_i| = 181440/48$ for $1 \le i \le 21$;

3. $|G_j^{(3)}| = 181440/2880$ for $1 \le j \le 21$;

Thus the number of polynomials in the 44 linearly equivalence classes is

$$1 \times 181440 + 21 \times \frac{181440}{48} + 21 \times \frac{181440}{2880} + 1 \times 1 = 4^9$$

which means

$$|\mathcal{G} \cup \{0\}| = |G_1^{(1)}| + \sum_{i=1}^{21} |G_i^{(2)}| + \sum_{j=1}^{21} |G_j^{(3)}| + |\{0\}|$$

Therefore $\mathfrak{P}$ is divided into

$$4^9 + 21 \times 6742 + 21 \times 274 + 43 + 1 = 409524$$

different linearly equivalence classes.

Table 1. Classification of $\mathfrak{P}$

	$\mathcal{A}_1 + \mathcal{G} \cup \{0\}$	$\mathcal{B}_i + \mathcal{G} \cup \{0\}$	$\mathcal{C}_i + \mathcal{G} \cup \{0\}$	$\mathcal{G} \cup \{0\}$
Cardinality	181440×4^9	3780×4^9	63×4^9	4^9
No. of classes	4^9	6742	274	44
$\lvert Stab(\Psi_1(f))\rvert$	1	48	2880	181440

In the above table the index i is between 1 and 21.

Table 2. Classification of $\mathcal{B}_i + \mathcal{G} \cup \{0\}(1 \le i \le 21)$

	$B_{i,j_1}^{(1)}$	$B_{i,1}^{(2)}$	$B_{i,j_3}^{(3)}$	$B_{i,j_4}^{(4)}$
No. of classes	160	1	2320	4261
$\lvert Stab(f)\rvert$	4	3	2	1
No. of g	12	16	24	48
No. of polys	45360	60480	90720	181440

5.2 Brief analysis of the results

We analyze the complexity of this example briefly as the argument for that the algorithms given in this paper is much more effective than the exhaustive search algorithm. The exhaustive search algorithm is straightforward, that is we randomly pick a polynomial from $\mathfrak{P}$ and let l run though all the invertible transformations, then we get one linearly equivalence class and remove it from $\mathfrak{P}$. We repeat this process until all polynomials in $\mathfrak{P}$ are considered.

Table 3. Classification of $\mathcal{C}_j + \mathcal{G} \cup \{0\} (1 \leq j \leq 21)$

	$C^{(1)}_{j,1}$	$C^{(2)}_{j,1}$	$C^{(3)}_{j,i_3}$	$C^{(4)}_{j,1}$	$C^{(5)}_{j,i_5}$
No. of classes	1	1	30	1	53
$\lvert Stab(f)\rvert$	480	288	96	60	48
No. of g	6	10	30	48	60
No. of polys	378	630	1890	3024	3780

	$C^{(6)}_{j,i_6}$	$C^{(7)}_{j,i_7}$	$C^{(8)}_{j,i_8}$	$C^{(9)}_{j,i_9}$	$C^{(10)}_{j,i_{10}}$
No. of classes	20	20	5	120	23
$\lvert Stab(f)\rvert$	10	6	4	2	1
No. of g	288	480	720	1440	2880
No. of polys	18144	30240	45360	90720	181440

From the experiment results we know that there are totally 409524 different linearly equivalence classes and $|GL_3(F_4)| = 181440$ which is the number of all invertible linear transformations, thus for the exhaustive search algorithm we have to do $409524 \times 181440 \approx 2^{36}$ operations. We note that one operation involves composition and comparison of two polynomials.

Then let us consider the algorithms given in this paper. In the first step, using algorithm 1, it is also exhaustive search algorithm, however, in this step we just consider the matrix equivalence classes, thus according to the result of step 1 we know that we have done $44 \times 181440 \approx 2^{23}$ operations. By theoretical analysis in step 2 we do nothing. In step 3, by algorithm 2 and result of step 1, we need $21 \times 6742 \times 48 \approx 2^{23}$ operations. In step 4, the process is similar to that in step 3, we need $21 \times 274 \times 2880 \approx 2^{24}$ operations. In the last step, it uses exhaustive search algorithm which needs $44 \times 181440 \approx 2^{23}$ operations by its result. Therefore the total operations needed for the algorithms are $2^{23} + 2^{23} + 2^{24} + 2^{23} \approx 2^{25.3}$ which is less than 2^{26}. By analysis above we get that the algorithms given in this paper are much more effective than the exhaustive search algorithm for this example.

There are some improvements in the algorithms given in this paper compared with the naive exhaustive search algorithm. In step 1 we use algorithm 1 which is actually an exhaustive search algorithm, but it is in the set of triangular polynomials, i.e. $\mathcal{F}$ as opposed to $\mathfrak{P}$. For $n_{\mathcal{F}} \ll n_{\mathfrak{P}}$, it is more feasible. And the theoretical analyses in step 2 save us the most time. In step 3 and 4, we use algorithm 2 in which the cycle for l is in a subgroup as opposed to all invertible linear transformations. Thus the two step are more effective. In the last step, similar to step 1, the exhaustive search is in $\mathcal{G} \cup \{0\}$ which is more feasible when q and n is small.

6 Conclusion

In this article, we present an efficient method to decide the exact number of equivalence classes in the case of IP1S problem for even characteristic ground field except F_2. This algorithm is far more efficient than the exhaustive search method. However the efficiency of the algorithm is not totally understood. More research will be needed.

References

1. Dongdai Lin, Jean-Charles Faugère, Ludovic Perret, and Tianze Wang. On enumeration of polynomial equivalence classes and their application to mpkc. *manuscript*, 2009.
2. T. Matsumoto and H. Imai. Public Quadratic Polynomial-tuples for efficient signature-verification and message-encryption. In *Eurocrypt*, volume 88, pages 419–453. Springer.
3. A.S. Fraenkel and Y. Yesha. Complexity of solving algebraic equations. *Information Processing Letters*, 10(4-5):178–179, 1980.
4. C. Wolf and B. Preneel. Large superfluous keys in multivariate quadratic asymmetric systems. *Proceedings of Public Key Cryptography-PKC 2005*, 3386:275–287.
5. V. Dubois, P.A. Fouque, and J. Stern. Cryptanalysis of sflash with slightly modified parameters. *Lecture Notes in Computer Science*, 4515:264, 2007.
6. V. Dubois, P. Fouque, A. Shamir, and J. Stern. Practical cryptanalysis of SFLASH. *LECTURE NOTES IN COMPUTER SCIENCE*, 4622:1, 2007.
7. C. Bouillaguet, P.A. Fouque, A. Joux, and J. Treger. A Family of Weak Keys in HFE (and the Corresponding Practical Key-Recovery).
8. J. Patarin. Hidden fields equations (HFE) and isomorphisms of polynomials (IP): Two new families of asymmetric algorithms. *Lecture Notes in Computer Science*, 1070:33–48, 1996.
9. A. Kipnis and A. Shamir. Cryptanalysis of the HFE public key cryptosystem by relinearization. *Lecture Notes in Computer Science*, pages 19–30, 1999.

Algebraic techniques for number field computations (extended abstract)

Jean-François Biasse[1], Michael J. Jacobson, Jr.[2]*, and Alan K. Silvester[3]

[1] École Polytechnique, 91128 Palaiseau, France
biasse@lix.polytechnique.fr
[2] Department of Computer Science, University of Calgary
2500 University Drive NW, Calgary, Alberta, Canada T2N 1N4
jacobs@cpsc.ucalgary.ca
[3] Department of Mathematics and Statistics, University of Calgary
2500 University Drive NW, Calgary, Alberta, Canada T2N 1N4
aksilves@math.ucalgary.ca

Abstract. We present improvements to the computations related to quadratic number fields and their application to cryptology.

1 Introduction

Quadratic number fields were proposed as a setting for public-key cryptosystems in the late 1980s by Buchmann and Williams [4, 5]. Their security relies on the hardness of the discrete logarithm problem in the imaginary case and the infrastructure discrete logarithm problem in the real case. The complexity of the algorithms for solving these problems is bounded by $L(1/2, O(1))$ where the subexponential function is defined as

$$L(\alpha, \beta) = e^{\beta \log |\Delta|^{\alpha} \log\log |\Delta|^{1-\alpha}},$$

where Δ is the discriminant of the order we are working with. This complexity is asymptotically slower than the one for factoring which reduces to the problem of computing the class number, and although the discrete logarithm problem in the Jacobian of elliptic curves remains exponential, there is no known reduction between this problem and the discrete logarithm problems in number fields either [14]. Therefore, studying the hardness of the discrete logarithm problem and of the principality testing problem on number fields is of cryptographic interest since they provide alternative cryptosystems whose security is unrelated to those currently being used.

Following the recommendations for securely choosing discriminants for use in quadratic field cryptography of [10] for the imaginary case and of [13] for the real case, we restricited our study to the case of prime discriminants. Indeed, in both imaginary and real cases, it usually suffices to use prime discriminants, as this forces the class number h_{Δ} to be odd. In the imaginary case, one then relies on

* The second author is supported in part by NSERC of Canada.

the Cohen-Lenstra heuristics [9] to guarantee that the class number is not smooth with high probability. In the real case, one uses the Cohen-Lenstra heuristics to guarantee that the class number is very small (and that the infrastructure is therefore large) with high probability. This restriction also prevents ourselves against the attacks described by Castagnos and Laguillaumie in the imaginary case [7] and by Castagnos, Joux, Laguillaumie and Nguyen in the real case [6] which are designed for discriminants of the form $\Delta = \pm np^2$.

In this paper, we describe improvements to the algorithms for computing the group structure of the ideal class group $Cl(\mathcal{O}_\Delta)$ of the maximal order $\mathcal{O}_\Delta$, solving instances of the discrete logarithm problem in $Cl(\mathcal{O}_\Delta)$, computing the regulator of $\mathcal{O}_\Delta$ when $\Delta > 0$ and solving the infrastructure discrete logarithm problem. After a brief description of the necessary background concerning number fields, we describe the last improvements affecting the linear algebra phase. We begin with a dedicated Gaussian elimination strategy to reduce the dimensions of the relation matrix M. Then, we provide numerical data about an implementation of a new algorithm for computing the Hermite Normal Form of M and thus deduce the group structure of $Cl(\mathcal{O}_\Delta)$. We also describe a new algorithm for the regulator computation, and finally we show the impact of a new algorithm due to Vollmer [23] for solving instances of the discrete logarithm problem in $Cl(\Delta)$.

Note that most of this paper is not new material. Section 3 is taken from [1], and Section 4 from [2]. Vollmer's algorithm described in Section 5 for computing the HNF was already used in [1], but it had not been compared with the existing HNF algorithm. Section 6 is taken from [3]. The security estimates given in Section 7 were not described before.

2 Number fields

Let $K = \mathbb{Q}(\sqrt{\Delta})$ be the quadratic field of discriminant Δ, where Δ is a non-zero integer congruent to 0 or 1 modulo 4 with Δ or $\Delta/4$ square-free. The integral closure of $\mathbb{Z}$ in K, called the maximal order, is denoted by $\mathcal{O}_\Delta$. An ideal can be represented by the two dimensional $\mathbb{Z}$-module

$$\mathfrak{a} = s\left[a\mathbb{Z} + \frac{b+\sqrt{\Delta}}{2}\mathbb{Z}\right] \quad ,$$

where $a, b, s \in \mathbb{Z}$ and $4a \mid b^2 - \Delta$. The integers a and s are unique, and b is defined modulo $2a$. The norm of $\mathfrak{a}$ is given by $\mathcal{N}(\mathfrak{a}) = as^2$. Ideals can be multiplied using Gauss' composition formulas for integral binary quadratic forms. Ideal norm respects this operation. The prime ideals of $\mathcal{O}_\Delta$ have the form $p\mathbb{Z} + (b_p + \sqrt{\Delta})/2\mathbb{Z}$ where p is a prime that is split or inert in K, i.e., the Kronecker symbol $(\Delta/p) \neq -1$. As $\mathcal{O}_\Delta$ is a Dedekind domain, every ideal can be factored uniquely as a product of prime ideals. We define the ideal class group as $Cl(\Delta) := \mathcal{I}_\Delta/\mathcal{P}_\Delta$, where $\mathcal{I}_\Delta$ is the set of invertible ideals, and $\mathcal{P}_\Delta$ is the set of principal ideals of $\mathcal{O}_\Delta$. This way, given two ideals $\mathfrak{a}$ and $\mathfrak{b}$ we have

$$[\mathfrak{a}] = [\mathfrak{b}] \in Cl(\Delta) \Longleftrightarrow \exists \alpha \in \mathbb{K} \ \mathfrak{b} = (\alpha)\mathfrak{a}.$$

$Cl(\Delta)$ is a finite group of cardinality h_Δ. To create M, we use sieving based techniques to find relations of the form

$$(\alpha) = \mathfrak{p}_1^{e_1} \dots \mathfrak{p}_n^{e_n},$$

where $\alpha \in \mathbb{K}$, and the $\mathfrak{p}_i$ are the prime ideals belonging to the set $\mathcal{B}$ of prime ideals of norm bounded by a certain bound B. Every time such a relation is found, we add the vector $[e_1, \dots, e_n]$ as a row of M. The most efficient way to do this is to use an adapation of the multiple polynomial quadratic sieve due to Jacobson [11], which was improved by Biasse [1] who used the large prime variants. Under the generalized Riemann hypothesis (GRH), if $B \geq 6 \log^2 |\Delta|$, the lattice Λ generated by all the possible relations satisfies

$$Cl(\Delta) \simeq \mathbb{Z}^n / \Lambda.$$

A linear algebra phase consisting in the computation of the Smith Normal Form (SNF) of M yields the group structure of $Cl(\Delta)$.

The units of $\mathcal{O}_\Delta$ form a multiplicative group

$$U_\Delta \simeq \mu_\Delta \times \{\varepsilon_\Delta\} \text{ (real) or } U_\Delta \simeq \mu_\Delta \text{ (imaginary)},$$

where μ_Δ are the roots of unity, and ε_Δ is the *fundamental unit*. In the real case, we compute the regulator $R_\Delta := \log |\varepsilon_\Delta|$ by finding kernel vectors of M during the linear algebra phase. Then, we give them as input to an algorithm due to Maurer [16] along with the generators α_i, $i \leq n$ of the relations. For cryptographic applications, we focus on solving the discrete logarithm problem (infrastructure DLP in the real case). Given two ideals $\mathfrak{a}$ and $\mathfrak{b}$ such that there exists $x \in \mathbb{Z}$ with $[\mathfrak{b}] = [\mathfrak{a}]^x \in Cl(\Delta)$, we aim at finding x and $\log |\alpha| \mod R_\Delta$ where $\mathfrak{b} = (\alpha)\mathfrak{a}^x$. In imaginary fields ($\Delta < 0$), α is trivial, and we only compute x.

3 Structured Gaussian elimination

Before applying the linear algebra algorithms we mentioned, we perform a Gaussian elimination step to reduce the dimensions of M. The main drawback of this strategy is that the density and the size of the coefficients of the matrix increase after each recombination of rows. We used a graph-based elimination strategy first described by Cavallar [8] for factorization, and then adapted by Biasse [1] to the context of number fields. Given a column involving N rows, this algorithm finds an optimal recombination strategy between the rows with respect to a cost function

$$C(r) := \sum_{1 \leq |e_i| \leq 8} 1 + 100 \sum_{|e_j| > 8} |e_j|,$$

where $r = [e_1, \dots, e_n]$ is a row. This cost function penalizes rows with large entries and high density. The first step of the algorithm is to build the complete graph $\mathcal{G}$ having N edges, and whose vertices (i, j) are weighted by the cost of the recombination involving the rows i and j according to C. Then, we compute the minimum spanning tree $\mathcal{T}$ of $\mathcal{G}$. Finally, we recombine the rows starting from those corresponding to the leaves of $\mathcal{T}$ and finishing with its root. At the end, we verify that the resulting matrix M_{red} has full rank with Linbox `rank` function. If not, we add more rows and repeat the process.

To illustrate the impact of this structured Gaussian elimination strategy over the naive Gaussian elimination, we monitored in Table 1 the evolution of the dimensions of the matrix, the average Hamming weight of its rows, the extremal values of its coefficients and the time taken for computing its HNF in the case of a relation matrix corresponding to $\Delta = 4(10^{60} + 3)$. We kept track of these values after all i-way merges for some values of i between 5 and 170. The original dimensions of the matrix were 2000×1700, and the timings were obtained on a 2.4 Ghz Opteron with 32GB of memory.

Table 1. Comparative table of elimination strategies

Naive Gauss						
i	Row Nb	Col Nb	Average weight	max coeff	min coeff	HNF time
5	1189	1067	27.9	14	-17	357.9
10	921	799	49.3	22	-19	184.8
30	757	635	112.7	51	-50	106.6
50	718	596	160.1	81	-91	93.7
70	699	577	186.3	116	-104	85.6
90	684	562	205.5	137	-90	79.0
125	664	542	249.0	140	-146	73.8
160	655	533	282.4	167	-155	72.0
170	654	532	286.4	167	-155	222.4
With dedicated elimination strategy						
i	Row Nb	Col Nb	Average weight	max coeff	min coeff	HNF time
5	1200	1078	26.8	13	-12	368.0
10	928	806	42.6	20	-15	187.2
30	746	624	82.5	33	-27	100.8
50	702	580	107.6	64	-37	84.3
70	672	550	136.6	304	-676	73.4
90	656	534	157.6	1278	-1088	67.5
125	637	515	187.1	3360	-2942	63.4
160	619	497	214.6	5324	-3560	56.9
170	615	493	247.1	36761280	-22009088	192.6

Table 1 shows that the use of our elimination strategy led to a matrix with smaller dimension (493 rows with our method, 533 with the naive elimination) and lower density (the average weight of its rows is of 214 with our method and 282 with the naive elimination). These differences result in an improvement of the time taken by the HNF computation: 56.9 with our method against 72.0 with the naive Gaussian elimination.

4 Regulator computation

To solve the infrastructure discrete logarithm problem, we first need to compute an approximation of the regulator. For this purpose, we used an improved version of Vollmer's system solving based algorithm [24] described by Biasse and Jacobson [2]. In order to find elements of the kernel, the algorithm creates extra relations r_i, $0 \leq i \leq k$ for some small integer k (in our experiments, we always have $k \leq 10$). Then, we solve the k linear systems $X_i M = r_i$ using the function `certSolveRedLong` from the IML library [22]. We augment M by adding the r_i as extra rows, and augment the vectors X_i with $k-1$ zero coefficients and -1 at index $n+i$, yielding

$$M' := \begin{pmatrix} M \\ \cdots\cdots\cdots \\ r_i \end{pmatrix}, \quad X_i' := \begin{pmatrix} X_i & \vdots & 0 \ldots 0 \; -1 \; 0 \ldots 0 \end{pmatrix} .$$

The X_i' are kernel vectors of M', which can be used along with the vector $\boldsymbol{v}$ containing the real parts of the relations, to compute a multiple of the regulator with Maurer's algorithm [17, Sec 12.1]. As shown in Vollmer [24], this multiple is equal to the regulator with high probability. In [2], it is shown that this method is faster than the one requiring a kernel basis because it only requires the solution to a few linear systems, and it can be adapted in such a way that the linear system involves M_{red}.

To illustrate the impact of this algorithm, we used the relation matrix obtained in the base case for discriminants of the form $4(10^n + 3)$ for n between 40 and 70. The timings are obtained on a 2.4GHz Opteron with 16GB of memory. In Table 2, the timings corresponding to our system solving approach are taken

Table 2. Comparative table of regulator computation time

n	Kernel Computation	System Solving
40	15.0	6.2
45	18.0	8.3
50	38.0	20.0
55	257.0	49.0
60	286.0	103.0
65	5009.0	336.0
70	10030.0	643.0

with seven kernel vectors. However, in most cases only two or three vectors are required to compute the regulator. As most of the time taken by our approach is spent on system solving, we see that computing fewer kernel vectors would result in an improvement of the timings, at the risk of obtaining a multiple of the regulator.

5 Class group computation

The class group structure is obtained with the diagonal coefficients of the SNF of M. Unfortunatelly, even after the Gaussian elimination, the dimensions of M_{red} are too large to allow a direct SNF computation. We thus have to compute the Hermite normal form (HNF) H of M first, and then to find the SNF of the essential part of H. A matrix H is said to be in HNF if with $\forall j < i : 0 \leq h_{ij} < h_{ii}$ and $\forall j > i : h_{ij} = 0$. For the imaginary case, we can use an algorithm due to Vollmer [25] which requires solutions to linear systems. For each $i \leq n$, we define two matrices

$$M_i = \begin{pmatrix} a_{1,1} & \dots & a_{m,1} \\ \vdots & & \vdots \\ a_{1,i} & \dots & a_{m,i} \end{pmatrix} \quad \text{and} \quad e_i = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}.$$

For each i, let h_i be the minimal denominator of a rational solution of the system $M_i x = e_i$ solved using the function `MinCertifiedSol` of IML [22]. We have $h_\Delta = \prod_i h_i$, and an extra computation involving modular reductions yields the essential part of the HNF of M. In most cases only a limited number of systems are to be soved. In the real case, we used a modular HNF algorithm [20]. It needs a multiple of h_Δ in input. To compute this multiple, we took the GCD of the determinants h_1 and h_2 of two $n \times n$ submatrices of M. We used the `determinant` function of Linbox for this purpose, which is why we refer to this strategy as NTL/Linbox in the following. Several implementations of an HNF algorithm are available today. In this section, we compare the most efficient ones: Magma, Sage, Kash, Pari to the methods we used in our computations. We used Magma V2.11-2 whereas a new algorithm is used since V.2.14. According to the developers' webpage, this algorithm should be more efficient on random dense matrices than the one we used, but we were not able to have it run on the same plateform. Sage Version 4.1.1, which is open-source, has an HNF algorithm based on the heuristic idea of Micciancio and Warinschi [18], which was analyzed and implemented By Pernet and Stein [21]. We used Kash Version 4 and Pari-2.3.5 whose HNF algorithm is due to Batut. Note here that Kash and Pari's algorithm provide the unimodular transformation matrix corresponding to the operations on the rows resulting in the HNF of the relation matrix. This step is not necessary for the ideal class group computation and can be time consuming.

Algorithm 1 Essential part of the HNF

```
Input: $\Delta$, relation matrix $A \in \mathbb{Z}^{m\times n}$ of full rank and $h_*$ such that $h^* \leq h < 2h^*$.
Output: The essential part of the HNF of $A$.
  $h \leftarrow 1$ , $i \leftarrow n$ , $l \leftarrow 1$.
  while $h < h_*$ do
    Compute the minimal denominator $h_i$ of a solution $\overrightarrow{v_i}$ of $A_i \cdot x = e_i$.
    $h \leftarrow h \cdot h_i$.
    $i \leftarrow i - 1$, $l \leftarrow l + 1$.
  end while
  Let $U$ be the $l \times m$ matrix whose rows are the $\overrightarrow{v_i}$ for $i \leq l$ and $H = (h_{ij})$ be the $l \times l$
  submatrix of $UA$ containing its last $l$ rows.
  for $2 \leq i \leq l$ do
    $h_{ij} \leftarrow h_{ij} \mod h_{ii}$ for all $j > i$.
  end for
  return $H$.
```

We also assessed the performances of Vollmer's algorithm on a single node, and on a several nodes. We noted between brackets the minimum number of nodes that is required to obtain the best performances in the parallelized version.

Table 3. Comparative timings of the HNF algorithms in the imaginary case

size	$\|\mathcal{B}\|$	algorithm	HNF time	stdev
130	320	Pari	16,64	9,01
		Kash	3,4	0,5
		Sage	5,6	0,68
		Magma	0,98	0,12
		NTL/Linbox	2,37	0,34
		Vollmer	2,2	1,04
		Vollmer Par (6)	0,61	0,05
150	400	Pari	30,56	4,09
		Kash	13,09	2,57
		Sage	12,69	8,06
		Magma	3,63	0,91
		NTL/Linbox	6,61	0,74
		Vollmer	7,12	3,54
		Vollmer Par (7)	1,62	0,11
160	450	Pari	54.41	25.55
		Kash	19,57	6,25
		Sage	19,96	5,26
		Magma	5,55	2,25
		NTL/Linbox	8,67	2,39
		Vollmer	8,02	2,62
		Vollmer Par (6)	1,91	0,38

In Table 3, we compared the time for computing the HNF of a relation matrix corresponding to negative discriminants of size ranging between 130 and 160 bits. For each discriminant size, we drew five random fundamental discriminants and computed a relation matrix on a 2.4 GHz Opteron with 8GB of memory. Unlike for the other benchmarks, we did not draw random prime discriminants because Vollmer's algorithm tends to be faster when working with relation matrices corresponding to cyclic ideal class groups. We notice that the best performances on a single are still optainded by Magma, but that the parallelized version of Vollmer's algorithm allows a significant speed-up if several nodes are available for this computation. This opens the way to fully parallelized algorithms since the relation collection phase is trivially parallelizable on as many nodes as we want.

6 DLP solving

For solving the discrete logarithm problem, we implemented an algorithm due to Vollmer [23] which also involves system solving. Given two ideals $\mathfrak{a}$ and $\mathfrak{b}$ such that $\mathfrak{b} = \mathfrak{a}^x$ for some integer x, it consists of finding two extra relations $\mathrm{r}_a : (\alpha_{\mathfrak{a}}) = \mathfrak{a} * \mathfrak{p}_1^{e_1} \dots \mathfrak{p}_n^{e_n}$ and $\mathrm{r}_b : (\alpha_{\mathfrak{b}}) = \mathfrak{b} * \mathfrak{p}_1^{f_1} \dots \mathfrak{p}_n^{f_n}$ and extending the factor base with two extra elements: $\mathcal{B}' = \mathcal{B} \cup \{\mathfrak{a}, \mathfrak{b}\}$. The extra relations are obtained by multiplying $\mathfrak{a}$ and $\mathfrak{b}$ by random power products of primes in $\mathcal{B}$ and sieving with the resulting ideal. Then, we construct the matrix

$$A' := \left(\begin{array}{c:c} A & (0) \\ \hdashline r_{\mathfrak{b}} & 1 \\ r_{\mathfrak{a}} & 0 \end{array}\right),$$

and solve the system $XA' = (0, \dots, 0, 1)$. The last coordinate of X necessarily equals $-x$. For each system, we used `certSolveRedLong` from the IML library [22]. It appeared that it was faster than both kernel computation and HNF computation. Testing the principality of an ideal I and finding α such that $(\alpha) = I$ can be done by finding a power product satisfying $I = \prod_i \mathfrak{p}_i^{e_i}$. Then, we need to solve the system $XM = b$ where $b = [e_1, \dots, e_n]$. If this system has a solution, then I is principal and its generator is $\alpha = \prod_i \alpha_i^{x_i}$ where the α_i are the generators of the relations used for constructing M and the x_i are the coefficients of X. An algorithm of Maurer [16] computes $\log |\alpha| \mod R_\Delta$ given R_Δ, $(\alpha_i)_{i \leq n}$ and X.

To study the impact of Vollmer's algorithm for solving the discrete logarithm problem without computing the structure of $Cl(\Delta)$, we provided numerical data in Table 4 for discriminants of size between 140 and 220 bits. The timings, given in CPU seconds, are averages of three different random prime discriminants, obtained with 2.4 GHz Opterons with 8GB or memory. We denote by "DL" the discrete logarithm computation using Vollmer's method and by "CL" the

Table 4. Comparison between class group computation and Vollmer's Algorithm

Size	Strategy	$\|\mathcal{B}\|$	Sieving	Elimination	Linear algebra	Total
140	CL	200	2.66	0.63	1.79	5.08
	DL	200	2.57	0.44	0.8	3.81
160	CL	300	11.77	1.04	8.20	21.01
	DL	350	10.17	0.73	2.75	13.65
180	CL	400	17.47	0.98	12.83	31.28
	DL	500	15.00	1.40	4.93	21.33
200	CL	800	158.27	7.82	81.84	247.93
	DL	1000	126.61	9.9	21.45	157.96
220	CL	1500	619.99	20.99	457.45	1098.43
	DL	1700	567.56	27.77	86.38	681.71

class group computation. We list the optimal factor base size for each algorithm and discriminant size (obtained via additional numerical experiments), the time for each of the main parts of the algorithm, and the total time. In all cases we allowed two large primes and took enough relations to ensure that M_{red} have full rank. Our results show that using Vollmer's algorithm for computing discrete logarithms is faster than the approach of [12] that also requires the class group.

7 Security estimates

As the relation collection clearly influences the overall time of the algorithm, we classified the quadratic discriminants with respect to the difficulty to create the relation matrix. During the sieving phase, we essentially test the smoothness of ideals with respect to $\mathcal{B}$. This boils down to testing the smoothness of norms with respect to primes p such that there is a $\mathfrak{p} \in \mathcal{B}$ satisfying $\mathcal{N}(\mathfrak{p}) = p$. Therefore, the hardest discriminants will be those satisfying $\left(\frac{\Delta}{p}\right) = -1$ for the small primes. When we choose discriminants at random, we cannot control this property, and we thus observe a high standard deviation in the performances of the DLP algorithms at a fixed discriminant size. To provide security estimates, we want to choose our instances of the discrete logarithm problem amongst the easiest ones, and ensure that the performances of the algorithm for solving the DLP are regular. In our experiments, we studied the performances of Vollmer's algorithm for solving the discrete logarithm problem on imaginary discriminants of three classes.

1. The *easy discrimiants*, satisfying $\left(\frac{\Delta}{p}\right) = 1$ for $p = 2, 3, 5, 7, 11$.
2. The *intermediate discriminants*, satisfying $\left(\frac{\Delta}{p}\right) = -1$ for $p = 2, 3, 5, 7, 11$ and $\left(\frac{\Delta}{p}\right) = 1$ for $p = 13, 17, 19, 23, 31$.
3. The *hard discrinants*, satisfying satisfying $\left(\frac{\Delta}{p}\right) = -1$ for $p \leq 31$.

In Table 5, we randomly drew 10 negative prime discriminants of size 170, 190 and 210 bits for each class of discriminant, and computed the time to solve an instance of the DLP. We used a 2,4 GHz Opteron with 32GB of memory and counted the time in CPU seconds.

Table 5. Comparative table of DLP time for $\Delta < 0$

	Easy		**Intermediate**		**Hard**	
Size	Average	Stdev	Average	Stdev	Average	Stdev
170	22.1	4.7	65.5	18.7	103.0	26.5
190	70.3	13.3	162.7	25.5	224.8	32.26
210	257.7	26.0	655.7	99.7	885.5	152.1

We observe in Table 5 that the time taken to solve the discrete logarithm problem corroborates the hypothesis we made on the difficulty of solving the discrete logarithm problem on the classes of discriminants we described. For our security estimates, we will take random discriminants belonging to the easy class, unlike in [3], where we took random discriminants and thus observed timings with large standard deviations. The rest of the methodology remains the same. We first provide timings allowing ourselves to decide if the run time of our algorithm follows the proven complexity $O(L_{|\Delta|}[1/2, 3\sqrt{2}/4 + o(1)]$, or the heuristic one $O(L_{|\Delta|}[1/2, 1 + o(1)])$. Then, we give the discriminant size required to provide an equivalent level of security as the RSA moduli recommended by NIST [19]. We assume that the run time of factoring algorithms follow the heuristic complexity of the generalized number field sieve $L_N[1/3, \sqrt[3]{64/9} + o(1)]$, and follow the approach of Hamdy and Möller [10] who used the equation

$$\frac{L_{N_1}[e, c]}{L_{N_2}[e, c]} = \frac{t_1}{t_2}, \tag{1}$$

to compute the run time t_2 on input size N_2, knowing the run time t_1 on input size N_1. To date, the largest RSA number factored is RSA-768, a 768 bit integer [15]. It is estimated in [15] that the total computation required 2000 2.2 GHz AMD Opteron years. As our computations were performed on a different architecture, we follow Hamdy and Möller and use the MIPS-year measurement to provide an architecture-neutral measurement. In this case, assuming that a 2.2 GHz AMD Opteron runs at 4400 MIPS, we estimate that this computation took 8.8×10^6 MIPS-years. Using this estimate in conjunction with (1) yields the estimated running times to factor RSA moduli of the sizes recommended by NIST given in Table 6, where we focus on the three classes of discriminants and compare them to random discriminants.

The results of our experiments for the imaginary case are given in Table 7, and for the real case in Table 8. They were obtained on 2.4 GHz Xeon with 2GB of memory. For each bit length of Δ, denoted by "size(Δ)," we list the

Table 6. Security Parameter Estimates

RSA	$\Delta < 0$ (rnd.)	$\Delta < 0$ (easy)	$\Delta < 0$ (int.)	$\Delta < 0$ (hard)	Est. time (MIPS-years)
768	640	661	640	631	8.80×10^6
1024	798	821	798	788	1.07×10^{10}
2048	1348	1378	1349	1337	1.25×10^{19}
3072	1827	1860	1827	1813	4.74×10^{25}
7680	3598	3643	3599	3579	1.06×10^{45}
15360	5971	6028	5972	5948	1.01×10^{65}
RSA	$\Delta > 0$ (rnd.)	$\Delta > 0$ (easy)	$\Delta > 0$ (int.)	$\Delta > 0$ (hard)	Est. time (MIPS-years)
768	634	638	632	629	8.80×10^6
1024	792	796	789	786	1.07×10^{10}
2048	1341	1346	1337	1334	1.25×10^{19}
3072	1818	1824	1814	1810	4.74×10^{25}
7680	3586	3594	3580	3575	1.06×10^{45}
15360	5957	5966	5949	5942	1.01×10^{65}

average time in seconds required to solve an instance of the appropriate discrete logarithm problem ($\overline{t_\Delta}$) and standard deviation (std). For each size, we solved 10 instances of both problems. In both cases, we concluded that the run timed was in $O(L_{|\Delta|}[1/2, 1 + o(1)])$.

References

1. J-F. Biasse, *Improvements in the computation of ideal class groups of imaginary quadratic number fields*, To appear in *Advances in Mathematics of Communications.*
2. J-F. Biasse and M. J. Jacobson, Jr., *Practical improvements to class group and regulator computation of real quadratic fields*, 2010, To appear in ANTS 9.
3. J-F. Biasse, M. J. Jacobson, Jr., and A. K. Silverster, *Security estimates for quadratic field based cryptosystems*, 2010, To appear in ACISP 2010.
4. J. Buchmann and H. C. Williams, *A key-exchange system based on imaginary quadratic fields*, Journal of Cryptology **1** (1988), 107–118.
5. ______, *A key-exchange system based on real quadratic fields*, CRYPTO '89, Lecture Notes in Computer Science, vol. 435, 1989, pp. 335–343.
6. G. Castagnos, A. Joux, F. Laguillaumie, and P. Q. Nguyen, *Factoring pq^2 with quadratic forms: Nice cryptanalyses*, ASIACRYPT '09: Proceedings of the 15th annual international conference on the theory and applications of cryptology and information security (Berlin, Heidelberg), Lecture Notes in Computer Science, vol. 5912, Springer-Verlag, 2009, pp. 469–486.
7. G. Castagnos and F. Laguillaumie, *On the security of cryptosystems with quadratic decryption: The nicest cryptanalysis*, EUROCRYPT '09: Proceedings of the 28th annual international conference on Advances in Cryptology (Berlin, Heidelberg), Lecture Notes in Computer Science, vol. 5479, Springer-Verlag, 2009, pp. 260–277.
8. S. Cavallar, *Strategies in filtering in the number field sieve*, ANTS-IV: Proceedings of the 4th International Symposium on Algorithmic Number Theory, Lecture Note in Computer Science, vol. 1838, Springer-Verlag, 2000, pp. 209–232.

Table 7. Average run times for the discrete logarithm problem in Cl_{Δ}, $\Delta < 0$

size(Δ)	$\overline{t_{\Delta}}$ (sec)	std	$L_{\|\Delta\|}[1/2, \sqrt{2}]/\overline{t_{\Delta}}$	$L_{\|\Delta\|}[1/2, 1]/\overline{t_{\Delta}}$
140	3.73	0.61	2.33×10^{12}	3.78×10^{8}
142	4.75	0.86	2.37×10^{12}	3.57×10^{8}
144	4.92	0.92	2.97×10^{12}	4.14×10^{8}
146	5.21	0.64	3.62×10^{12}	4.68×10^{8}
148	5.92	0.69	4.10×10^{12}	4.93×10^{8}
150	6.36	1.47	4.92×10^{12}	5.49×10^{8}
152	6.65	0.89	6.05×10^{12}	6.26×10^{8}
154	8.20	1.27	6.30×10^{12}	6.06×10^{8}
156	10.68	4.11	6.20×10^{12}	5.55×10^{8}
158	10.36	2.23	8.18×10^{12}	6.81×10^{8}
160	12.11	2.51	8.95×10^{12}	6.93×10^{8}
162	18.03	3.64	7.67×10^{12}	5.53×10^{8}
164	22.78	8.03	7.74×10^{12}	5.20×10^{8}
166	21.23	4.84	10.58×10^{12}	6.62×10^{8}
168	26.59	8.32	10.75×10^{12}	6.27×10^{8}
170	29.15	8.15	12.45×10^{12}	6.77×10^{8}
172	32.24	6.78	14.28×10^{12}	7.24×10^{8}
174	49.71	18.65	11.74×10^{12}	5.55×10^{8}
176	52.08	12.57	14.18×10^{12}	6.26×10^{8}
178	51.99	9.79	17.96×10^{12}	7.40×10^{8}
180	75.10	18.75	15.70×10^{12}	6.04×10^{8}
182	73.34	4.76	2.02×10^{13}	7.29×10^{8}
184	80.66	14.19	2.32×10^{13}	7.81×10^{8}
186	79.69	16.25	2.96×10^{13}	9.30×10^{8}
188	93.73	11.09	3.16×10^{13}	9.30×10^{8}
190	100.89	13.93	3.69×10^{13}	10.15×10^{8}
192	117.18	14.71	3.98×10^{13}	10.26×10^{8}
194	133.77	16.43	4.37×10^{13}	10.54×10^{8}
196	167.70	21.11	4.37×10^{13}	9.86×10^{8}
198	162.21	13.59	5.65×10^{13}	11.94×10^{8}
200	195.29	24.69	5.87×10^{13}	11.61×10^{8}
202	291.58	27.96	4.90×10^{13}	9.10×10^{8}
204	292.70	42.55	6.09×10^{13}	10.59×10^{8}
206	335.39	39.38	6.63×10^{13}	10.80×10^{8}
208	360.00	51.24	7.69×10^{13}	11.75×10^{8}
210	396.10	82.10	8.69×10^{13}	12.46×10^{8}
212	448.85	72.62	9.53×10^{13}	12.82×10^{8}
214	535.67	123.40	9.92×10^{13}	12.52×10^{8}
216	595.56	109.94	11.07×10^{13}	13.12×10^{8}
218	641.99	89.52	12.73×10^{13}	14.16×10^{8}
220	829.98	151.75	12.19×10^{13}	12.74×10^{8}
230	1564.74	226.924	18.60×10^{13}	14.27×10^{8}
240	1564.74	226.924	52.48×10^{13}	29.71×10^{8}
250	5552.59	953.788	40.94×10^{13}	17.20×10^{8}

Table 8. Average run times for the infrastructure discrete logarithm problem.

size(Δ)	$\overline{t_\Delta}$ (sec)	std	$L_{\lvert\Delta\rvert}[1/2, \sqrt{2}]/\overline{t_\Delta}$	$L_{\lvert\Delta\rvert}[1/2, 1]/\overline{t_\Delta}$
140	3.66	3.00	2.38×10^{12}	3.86×10^{8}
142	9.33	0.85	1.21×10^{12}	1.82×10^{8}
144	10.49	1.00	1.39×10^{12}	1.94×10^{8}
146	10.78	0.92	1.75×10^{12}	2.26×10^{8}
148	10.21	1.32	2.38×10^{12}	2.86×10^{8}
150	11.14	1.70	2.81×10^{12}	3.13×10^{8}
152	12.29	1.39	3.27×10^{12}	3.39×10^{8}
154	11.11	0.97	4.65×10^{12}	4.47×10^{8}
156	14.58	2.59	4.54×10^{12}	4.06×10^{8}
158	15.46	2.35	5.48×10^{12}	4.56×10^{8}
160	15.72	2.21	6.89×10^{12}	5.34×10^{8}
162	29.48	6.72	4.69×10^{12}	3.38×10^{8}
164	31.71	3.49	5.56×10^{12}	3.73×10^{8}
166	33.82	4.54	6.64×10^{12}	4.15×10^{8}
168	37.61	4.95	7.60×10^{12}	4.43×10^{8}
170	40.06	5.43	9.06×10^{12}	4.92×10^{8}
172	42.63	5.80	10.80×10^{12}	5.48×10^{8}
174	47.45	8.81	12.30×10^{12}	5.82×10^{8}
176	50.73	8.92	14.56×10^{12}	6.43×10^{8}
178	55.09	14.07	16.95×10^{12}	6.99×10^{8}
180	65.12	25.86	18.11×10^{12}	6.97×10^{8}
182	218.06	23.48	6.82×10^{12}	2.45×10^{8}
184	204.61	18.27	9.16×10^{12}	3.08×10^{8}
186	222.69	21.26	10.59×10^{12}	3.33×10^{8}
188	220.46	22.92	13.45×10^{12}	3.95×10^{8}
190	221.67	24.60	16.80×10^{12}	4.62×10^{8}
192	232.10	27.68	2.01×10^{13}	5.18×10^{8}
194	239.50	29.81	2.44×10^{13}	5.89×10^{8}
196	307.33	38.90	2.38×10^{13}	5.38×10^{8}
198	298.28	55.29	3.07×10^{13}	6.49×10^{8}
200	337.96	73.80	3.39×10^{13}	6.71×10^{8}
202	791.08	113.13	1.80×10^{13}	3.35×10^{8}
204	888.10	95.55	2.01×10^{13}	3.49×10^{8}
206	900.51	61.40	2.47×10^{13}	4.02×10^{8}
208	871.15	80.96	3.17×10^{13}	4.85×10^{8}
210	948.95	114.40	3.63×10^{13}	5.20×10^{8}
212	1021.10	79.65	4.19×10^{13}	5.63×10^{8}
214	1091.83	160.53	4.86×10^{13}	6.14×10^{8}
216	1110.52	146.59	5.93×10^{13}	7.03×10^{8}
218	1250.34	194.58	6.53×10^{13}	7.27×10^{8}
220	1415.05	237.89	7.15×10^{13}	7.47×10^{8}
230	4196.60	812.71	6.93×10^{13}	5.32×10^{8}
240	6409.90	1097.76	12.81×10^{13}	7.25×10^{8}
250	16253.60	2653.25	13.98×10^{13}	5.87×10^{8}

9. H. Cohen and H. W. Lenstra, Jr., *Heuristics on class groups of number fields*, Number Theory, Lecture notes in Math., vol. 1068, Springer-Verlag, New York, 1983, pp. 33–62.
10. S. Hamdy and B. Möller, *Security of cryptosystems based on class groups of imaginary quadratic orders*, Advances in Cryptology - ASIACRYPT 2000, Lecture Notes in Computer Science, vol. 1976, 2000, pp. 234–247.
11. M. J. Jacobson, Jr., *Subexponential class group computation in quadratic orders*, Ph.D. thesis, Technische Universitt Darmstadt, Darmstadt, Germany, 1999.
12. ———, *Computing discrete logarithms in quadratic orders*, Journal of Cryptology **13** (2000), 473–492.
13. M. J. Jacobson, Jr., R. Scheidler, and H. C. Williams, *The efficiency and security of a real quadratic field based key exchange protocol*, Public-Key Cryptography and Computational Number Theory (Warsaw, Poland), de Gruyter, 2001, pp. 89–112.
14. M. J. Jacobson, Jr. and H. C. Williams, *Solving the Pell equation*, CMS Books in Mathematics, Springer-Verlag, 2009, ISBN 978-0-387-84922-5.
15. T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, H. te Riele, A. Timofeev, and P. Zimmerman, *Factorization of a 768-bit RSA modulus*, Eprint archive no. 2010/006, 2010.
16. M. Maurer, *Regulator approximation and fundamental unit computation for real quadratic orders*, Ph.D. thesis, Technische Universitt Darmstadt, Darmstadt, Germany, 1999.
17. ———, *Regulator approximation and fundamental unit computation for real-quadratic orders*, Ph.D. thesis, Technische Universität Darmstadt, Darmstadt, Germany, 2000.
18. D. Micciancio and B. Warinschi, *A linear space algorithm for computing the hermite normal form*, ISSAC '01: Proceedings of the 2001 international symposium on Symbolic and algebraic computation (New York, NY, USA), ACM, 2001, pp. 231–236.
19. National Institute of Standards and Technology (NIST), *Recommendation for key management - part 1: General (revised)*, NIST Special Publication 800-57, March, 2007, See: `http://csrc.nist.gov/groups/ST/toolkit/documents/SP800-57Part1_3-8-07.pdf`.
20. R. Kannan P. Domich and L. Trotter, *Hermite normal form computation using modulo determinant arithmetic*, Math. Oper. Research **12** (1987), 50–59.
21. C. Pernet and W. Stein, *Fast computation of hermite normal forms of random integer matrices*, Journal of Number Theory **In Press, Corrected Proof** (2010), –.
22. A. Storjohann, *Iml*, http://www.cs.uwaterloo.ca/∼astorjoh/iml.html.
23. U. Vollmer, *Asymptotically fast discrete logarithms in quadratic number fields*, Algorithmic Number Theory — ANTS-IV, Lecture Notes in Computer Science, vol. 1838, 2000, pp. 581–594.
24. ———, *An accelerated Buchmann algorithm for regulator computation in real quadratic fields*, Algorithmic Number Theory — ANTS-V, Lecture Notes in Computer Science, vol. 2369, 2002, pp. 148–162.
25. Ulrich Vollmer, *A note on the Hermite basis computation of large integer matrices*, International Symposium on Symbolic and Algebraic Computation, ISSAC '03 (J. Rafael Sendra, ed.), ACM Press, 2003, pp. 255–257.

Implicit Factoring with Shared Most Significant and Middle Bits

Jean-Charles Faugère, Raphaël Marinier, and Guénaël Renault

UPMC, Université Paris 06, LIP6
INRIA, Centre Paris-Rocquencourt, SALSA Project-team
CNRS, UMR 7606, LIP6
4, place Jussieu
75252 Paris, Cedex 5, France
jean-charles.faugere@inria.fr, raphael.marinier@polytechnique.edu,
guenael.renault@lip6.fr

Keywords: implicit factorization, lattices, RSA

The corresponding paper version of this extended abstract is accepted for PKC2010 [3]

The problem of factoring integers given additional information about their factors has been studied since 1985. In [6], Rivest and Shamir showed that $N = pq$ of bit-size n and with balanced factors ($\log_2(p) \approx \log_2(q) \approx \frac{n}{2}$) can be factored in polynomial time as soon as we have access to an *oracle* that returns the $\frac{n}{3}$ most significant bits (MSBs) of p. Beyond its theoretical interest, the motivation behind this is mostly of cryptographic nature. In fact, during an attack of an RSA-encrypted exchange, the cryptanalyst may have access to additional information beyond the RSA public parameters (e, N), that may be gained for instance through side-channel attacks revealing some of the bits of the secret factors. Besides, some variations of the RSA Cryptosystem purposely leak some of the secret bits (for instance, [8]). In 1996, Rivest and Shamir's results were improved in [2] by Coppersmith applying lattice-based methods to the problem of finding small integer roots of bivariate integer polynomials (the now so-called *Coppersmith's method*). It requires only half of the most significant bits of p to be known to the cryptanalyst (that is $\frac{n}{4}$).

In PKC 2009, May and Ritzenhofen [5] significantly reduced the power of the oracle. Given an RSA modulus $N_1 = p_1q_1$, they allow the oracle to output a new and different RSA modulus $N_2 = p_2q_2$ such that p_1 and p_2 share at least t least significant bits (LSBs). Note that the additional information here is only *implicit*: the attacker does not know the actual value of the t least significant bits of the p_i's, he only knows that p_1 and p_2 share them. In the rest of the paper, we will refer to this problem as the problem of *implicit factoring*. When q_1 and q_2 are α-bit primes, May and Ritzenhofen's lattice-based method rigorously finds in quadratic time the factorization of N_1 and N_2 when $t \geq 2\alpha + 3$. Besides, their technique heuristically generalizes to $k-1$ oracle queries that give access to k different RSA moduli $N_i = p_iq_i$ with all the p_i's sharing t least significant bits. With $k-1$ queries the bound on t improves to: $t \geq \frac{k}{k-1}\alpha$. Note that these results are of interest for unbalanced RSA moduli: for instance, if $N_1 = p_1q_1$, $N_2 = p_2q_2$ are 1000-bit RSA moduli and the q_i's are 200-bit primes, knowing that p_1 and p_2 share at least 403 least significant bits out of 800 is enough to factorize N_1 and N_2 in polynomial time. Note also that the method absolutely requires that the shared bits be the least significant ones. They finally apply their method to factorize k n-bit balanced RSA moduli $N_i = p_iq_i$ under some conditions and with an additional exhaustive search of $2^{\frac{n}{4}}$.

Very recently, in [7], Sarkar and Maitra applied Coppersmith and Gröbner-basis techniques on the problem of implicit factoring, and improved heuristically the bounds in some of the cases. Contrary to

[5], their method applies when either (or both) LSBs or MSBs of p_1, p_2 are shared (or when bits in the middle are shared). Namely, in the case of shared LSBs they obtain better theoretical bounds on t than [5] as soon as $\alpha \geq 0.266n$. Besides, their experiments often perform better than their theoretical bounds, and they improve in practice the bound on t of [5] when $\alpha \geq 0.21n$. Note finally that their bounds are very similar in the two cases of shared MSBs and shared LSBs. Readers interested in getting their precise bounds may refer to their paper [7].

Unfortunately, Sarkar and Maitra's method is heuristic even in the case of two RSA moduli, and does not generalize to $k \geq 3$ RSA moduli. In fact, when the p_i's share MSBs and/or LSBs, their method consists in building a polynomial f_1 in three variables, whose roots are $(q_2+1, q_1, \frac{p_1-p_2}{2^\gamma})$, where γ is the number of shared LSBs between p_1 and p_2. That is, $\frac{p_1-p_2}{2^\gamma}$ represents the part of $p_1 - p_2$ where the shared bits do not cancel out. To find the integer roots of f_1, they use the Coppersmith-like technique of [4] which consists in computing two (or more) new polynomials $f_2, f_3, \ldots$ sharing the same roots as f_1. If the variety defined by $f_1, f_2, f_3, \ldots$ is 0-dimensional, then the roots can be easily recovered computing resultants or Gröbner basis. However, with an input polynomial with more than two variables, the method is heuristic: there is no guarantee for the polynomials $f_1, f_2, f_3, \ldots$ to define a 0-dimensional variety. We reproduced the results of Sarkar and Maitra and we observed that $f_1, f_2, f_3, \ldots$ almost never defined a 0-dimensional variety. They observed however that it was possible to recover the roots of the polynomials directly by looking at the coefficients of the polynomials in the Gröbner basis of the ideal generated by the f_i's, even when the ideal was of positive dimension. The assumption on which their work relies is that it will always be possible. For instance, in the case of shared MSBs between p_1 and p_2, they found in their experiments that the Gröbner basis contained a polynomial multiple of $x - \frac{q_2}{q_1}y - 1$ whose coefficients lead immediately to the factorization of N_1 and N_2. They support their assumption by experimental data: in most cases their experiments perform better than their theoretical bounds. It seems nevertheless that their assumption is not fully understood.

Our contribution consists of a novel and rigorous lattice-based method that address the implicit factoring problem when p_1 and p_2 share *most* significant bits. That is, we obtained an analog of May and Ritzenhofen's results for shared MSBs, and our method is rigorous contrary to the work of Sarkar and Maitra in [7]. Namely, let $N_1 = p_1q_1$ and $N_2 = p_2q_2$ be two RSA moduli of same bit-size n. If q_1, q_2 are α-bit primes and p_1, p_2 share t most significant bits, our method provably factorizes N_1 and N_2 as soon as $t \geq 2\alpha + 3$ (which is the same as the bound on t for least significant bits in [5]). This is the first rigorous bound on t when p_1 and p_2 share most significant bits. From this method, we deduce a new heuristic lattice-based for the case when p_1 and p_2 share t bits in the middle. Moreover, contrary to [7], these methods heuristically generalize to an arbitrary number k of RSA moduli and do not depend on the position of the shared bits in the middle, allowing us to factorize k RSA moduli as soon as $t \geq \frac{k}{k-1}\alpha + 6$ (resp. $t \geq \frac{2k}{k-1}\alpha + 7$) most significant bits (resp. bits in the middle) are shared between the p_i's (more precise bounds are stated later in this paper). A summary of the comparison of our method with the methods in [5] and [7] can be found in table 1.

Let's give the main idea of our method with 2 RSA moduli in the case of shared MSB's. Consider the lattice L spanned by the row vectors $\mathbf{v_1}$ and $\mathbf{v_2}$ of the following matrix:

$$\begin{pmatrix} K & 0 & N_2 \\ 0 & K & -N_1 \end{pmatrix} \quad \text{where } K = \lfloor 2^{n-t+\frac{1}{2}} \rfloor$$

Consider also the following vector in L:

$$\mathbf{v_0} = q_1\mathbf{v_1} + q_2\mathbf{v_2} = (q_1K, q_2K, q_1q_2(p_2 - p_1))$$

The key observation is that the t shared significant bits of p_1 and p_2 cancel out in the algebraic relation $q_1N_2 - q_2N_1 = q_1q_2(p_2 - p_1)$. Furthermore, we choose K in order to force the coefficients of a shortest

Table 1: Comparison of our results against the results of [5] and [7]

k (number of RSA moduli)	May, Ritzenhofen's Results [5]	Sarkar, Maitra's Results [7]	Our results
$k = 2$	When p_1, p_2 share t LSBs: rigorous bound of $t \geq 2\alpha + 3$ using 2-dimensional lattices of $\mathbb{Z}^2$.	When p_1, p_2 share either t LSBs or MSBs: heuristic bound better than $t \geq 2\alpha + 3$ when $\alpha \geq 0.266n$, and experimentally better when $\alpha \geq 0.21n$. In the case of t shared bits in the middle, better bound than $t \geq 4\alpha + 7$ but depending on the position of the shared bits. Using 46-dimensional lattices of $\mathbb{Z}^{46}$	When p_1, p_2 share t MSBs: rigorous bound of $t \geq 2\alpha + 3$ using 2-dimensional lattices of $\mathbb{Z}^3$. In the case of t bits shared in the middle: heuristic bound of $t \geq 4\alpha + 7$ using 3-dimensional lattices of $\mathbb{Z}^3$.
$k \geq 3$	When the p_i's all share t LSBs: heuristic bound of $t \geq \frac{k}{k-1}\alpha$ using k-dimensional lattices of $\mathbb{Z}^k$.	Cannot be directly applied.	When the p_i's all share t MSBs (resp. bits in the middle): heuristic bound of $t \geq \frac{k}{k-1}\alpha + \delta_k$ (resp. $t \geq \frac{2k}{k-1}\alpha + \delta_k$), with $\delta_k \leq 6$ (resp. ≤ 7) and using k-dimensional ($\frac{k(k+1)}{2}$-dimensional) lattices of $\mathbb{Z}^{\frac{k(k+1)}{2}}$.

vector of L on the basis $(\mathbf{v_1}, \mathbf{v_2})$ to be of the order of $2^\alpha \approx q_1 \approx q_2$. We proved a result stating that $\mathbf{v_0}$ is indeed a shortest vector of L (thus N_1 and N_2 can be factored in polynomial time) as soon as $t \geq 2\alpha + 3$. Besides, we generalized this construction to an arbitrary number of k RSA moduli such that a small vector of the lattice harnesses the same algebraic relation, and to shared middle bits. However, the generalized constructions in both cases become heuristic: we use the Gaussian heuristic to find a condition on t for this vector to be a shortest of the lattice. More precisely, we obtained the following results

Theorem 1. *Let $N_1 = p_1q_1, N_2 = p_2q_2$ be two n-bit RSA moduli, where the q_i's are α-bit primes and the p_i's are primes that share t most significant bits. If $t \geq 2\alpha + 3$, then N_1 and N_2 can be factored in quadratic time in n.*

Let $\mathscr{C}(k, s, B)$ be the time to find a shortest vector of a k-dimensional lattice of $\mathbb{Z}^s$ given by B-bit basis vectors. We have the following generalization and application which are **stated under Gaussian heuristic** (we assume that if $\pm\mathbf{v_0}$ is a vector of a d-dimensional lattice L with norm smaller than $\sqrt{\frac{d}{2\pi e}}\mathrm{Vol}(L)^{\frac{1}{d}}$ then it is a shortest vector of L):

Theorem 2. *Let $N_1 = p_1q_1, \ldots, N_k = p_kq_k$ be k n-bit RSA moduli, with the q_i's being α-bit primes, and the p_i's being primes that all share t most significant bits. The N_i's can be factored in time $\mathscr{C}(k, \frac{k(k+1)}{2}, n)$, as soon as*

$$t \geq \frac{k}{k-1}\alpha + 1 + \frac{k}{2(k-1)}\left(2 + \frac{\log_2(k)}{k} + \log_2(\pi e)\right)$$

Theorem 3. *Let $N_1 = p_1q_1, \ldots, N_k = p_kq_k$ be k n-bit RSA moduli, where the q_i's are α-bit primes and the p_i's are primes that all share t bits from the position t_1 to $t_2 = t_1 + t$. The N_i's can be factored in time $\mathscr{C}(\frac{k(k+1)}{2}, \frac{k(k+1)}{2}, n)$, as soon as*

$$t \geq 2\alpha + \frac{2}{k-1}\alpha + \frac{k+1}{2(k-1)}\log_2(2\pi e)$$

We support these results by experimental facts. In order to check the validity of Gaussian heuristic in our case and the quality of our bounds on t, we implemented the methods on Magma 2.15 [1].

The MSB case. We generated many random 1024-bit RSA moduli, for various values of α and t. We observed that the results were similar for other values of n. In the case where $k = 2$, we used the Lagrange reduction to find with certainty a shortest vector of the lattice, and for $3 \leq k \leq 40$ we compared Schnorr-Euchner's algorithm (that provably outputs a shortest vector of the lattice) with LLL (that gives an exponential approximation of a shortest vector). We used only LLL for $k = 80$.

We conducted experiments for $k = 2, 3, 10, 40$ and 80, and for several values for α. In the rigorous case $k = 2$, we observed that the attack consistently goes one bit further with 100% success rate than our bound in Theorem 1. In all our experiments concerning the heuristic cases $k \geq 3$, we observed that we had 100% success rate (thus, Gaussian heuristic was always true in our case) when t was within the bound of Theorem 2. That means that the assumption concerning the Gaussion heuristic was always true in our experiments. Moreover, we were often able to go a few bits (up to 3) beyond the theoretical bound on t. When the success rate was not 100% (that is, beyond our experimental bounds on t), we found that Gaussian heuristic was not true in a very limited number of the cases (less than 3%). Finally, up to dimension 80, LLL was always sufficient to find $\mathbf{v_0}$ when t was within the bound of Theorem 2, and Schnorr-Euchner's algorithm allowed us to go one bit further than LLL in dimension 40.

The middle bits case. Contrary to the case of shared MSBs, Gaussian heuristic may fail when we apply our method with shared bits in the middle since there may exist some exceptional shortest vectors which does not correspond to the solution of our problem. When $k = 2$ the phenomenon of exceptional short vectors rarely appeared when t was within the bound of Theorem 3 (less than 1% of failure and did not depend on the position of the bits, moreover, we were generally allowed to go 2 or 3 bits further with 90% of success). When $k \geq 3$ it was not still the case. When Schnorr-Euchner's algorithm did not return $\mathbf{v_0}$, we tried to find it in a reduced basis computed by LLL. Our experiments showed that for the same size of problems the rate of success is approximately 80% when t was within the bound of Theorem 3 and allowed us to go one or two bits further with success rate $\approx$50%.

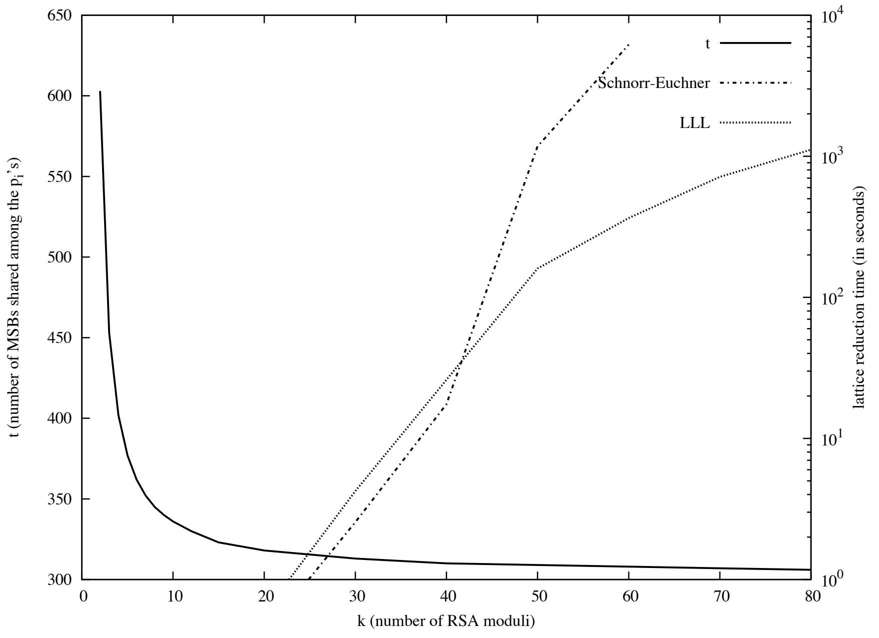
Efficiency comparisons. Additionally, we show in Table 2 the lowest value of t with 100% success rate and the running-time of LLL and Schnorr-Euchner's algorithm for several values of k (k RSA moduli with p_i's factors sharing t MSBs). For each k, we show the worst running-time we encountered when running 10 tests on an Intel Xeon E5420 at 2.5Ghz. We see that all individual tests completed in less than 1 second for $2 \leq k \leq 20$. We used Schnorr-Euchner's algorithm up to $k = 60$ where it took at most 6200 seconds. LLL completes under one minute for $20 \leq k \leq 40$ and in less than 30 minutes for $40 \leq k \leq 80$.

Applications of implicit factoring have not yet been extensively studied, and we believe that they will develop. The introduction of [5] gives some ideas for possible applications. They include destructive applications with malicious manipulation of public key generators, as well as possibly constructive ones. Indeed, our work shows that when $t \geq 2\alpha + 3$, it is as hard to factorize $N_1 = p_1 q_1$, as generating $N_2 = p_2 q_2$ with p_2 sharing t most significant bits with p_1. This problem could form the basis of a cryptographic primitive.

References

1. Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system I: The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).
2. Don Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In Ueli M. Maurer, editor, *EUROCRYPT*, volume 1070 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 1996.

Table 2: Running time of LLL and Schnorr-Euchner's algorithm, and bound on t as k grows. (Shared MSBs with $\alpha = 300$ and $n = 1024$)



3. Jean-Charles Faugère, Raphaël Marinier, and Guénaël Renault. Implicit factoring with shared most significant and middle bits. In P.Q. Nguyen and D. Poincheval, editors, *PKC*, volume 6056 of *Lecture Notes in Computer Science*, pages 70–87. Springer-Verlag, 2010.
4. Ellen Jochemsz and Alexander May. A strategy for finding roots of multivariate polynomials with new applications in attacking rsa variants. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 267–282. Springer, 2006.
5. Alexander May and Maike Ritzenhofen. Implicit factoring: On polynomial time factoring given only an implicit hint. In Stanislaw Jarecki and Gene Tsudik, editors, *Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2009.
6. Ronald L. Rivest and Adi Shamir. Efficient factoring based on partial information. In Franz Pichler, editor, *EUROCRYPT*, volume 219 of *Lecture Notes in Computer Science*, pages 31–34. Springer, 1985.
7. Santanu Sarkar and Subhamoy Maitra. Further Results on Implicit Factoring in Polynomial Time. *Advances in Mathematics of Communications*, 3(2):205–217, 2009.
8. Scott A. Vanstone and Robert J. Zuccherato. Short rsa keys and their generation. *J. Cryptology*, 8(2):101–114, 1995.

On the Immunity of Boolean functions Against Probabilistic Algebraic Attacks [⋆] (Extended Abstract)

Meicheng Liu[1,2] and Dongdai Lin[1]

[1] The State Key Laboratory of Information Security, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
[2] Graduate University of Chinese Academy of Sciences, Beijing 100049, China
meicheng.liu@gmail.com, ddlin@is.iscas.ac.cn

Abstract. In this paper, we study the immunity of Boolean functions against probabilistic algebraic attacks and its relations with other cryptographic properties. To measure the ability of Boolean functions resistant to probabilistic algebraic attacks, we introduce the notions of distance to algebraic immunity and k-error algebraic immunity, and then present the bounds of distance to algebraic immunity. Besides, new relations between high order nonlinearity and algebraic immunity are obtained.

1 Introduction

Algebraic attacks, which use cleverly over-defined systems of multi-variable nonlinear equations to recover the secret key, have been regarded as a great threat against stream ciphers based on LFSR. A new cryptographic property known as algebraic immunity(AI), used to scale the ability of Boolean functions to resist algebraic attacks, is proposed for Boolean functions. There are a large number of literatures investigating algebraic immunity but few referring to the immunity of Boolean functions against probabilistic algebraic attacks. As long ago as 2003, Courtois and Meier[2] described the probabilistic scenario of algebraic attacks:

S4. There exists a nonzero function g of low degree such that gf is equal to a function of low degree with probability $1-\varepsilon$.

Later, Braeken and Preneel[1] generalized S4 to the two scenarios S4a and S4b:

S4a. There exists a nonzero function g of low degree such that $gf = g$ on 0_f with probability $1-\varepsilon$, where $0_f = \{x|f(x) = 0\}$.

S4b. There exists a nonzero function g of low degree such that $gf = 0$ on 1_f with probability $1-\varepsilon$, where $1_f = \{x|f(x) = 1\}$.

Recently, Pometun [4] introduced the notion of the high order partial nonlinearity as a measure of the ability of Boolean functions resistant to probabilistic algebraic attacks, and obtained several properties of the high order partial nonlinearity especially for balanced functions.

[⋆] This work was supported by the National Natural Science Foundation of China under Grants 10971246 and 60970152.

Definition 1 *[4, Definition 9] The partial nonlinearity of the r-th order of the Boolean function f is given by*

$$\mathrm{nlp}_r(f) = \min_{c\in\mathbb{F}_2, 1\le \deg(g)\le r} 2^n \Pr[g \ne f | f = c].$$

In this paper, we further research the immunity of Boolean functions against probabilistic algebraic attacks. First, we unify the two scenarios S4a and S4b into one scenario, and then study the validity of probabilistic algebraic attacks. Furthermore, we introduce the notions of distance to algebraic immunity (similar to but different to high order partial nonlinearity) and k-error algebraic immunity, and present the relations among distance to algebraic immunity, k-error algebraic immunity, algebraic immunity and high order nonlinearity.

2 Preliminary

Let $\mathbb{F}_2$ be the binary field. An n-variable Boolean function is a mapping from $\mathbb{F}_2^n$ into $\mathbb{F}_2$. Denote the set of all n-variable Boolean functions by $\mathbf{B}_n$. An n-variable Boolean function can be uniquely represented as a truth table of length 2^n,

$$f = [f(0,0,\cdots,0), f(1,0,\cdots,0),\cdots, f(1,1,\cdots,1)].$$

The number of ones in the truth table of f is called the Hamming weight of f, denoted by $\mathrm{wt}(f)$. If $\mathrm{wt}(f) = 2^{n-1}$, then f is called balanced. The number of $x \in \mathbb{F}_2^n$ at which $f(x) \ne g(x)$ is called the Hamming distance between f and g , denoted by $\mathrm{d}(f,g)$. It's well known that $\mathrm{d}(f,g) = \mathrm{wt}(f+g)$.

An n-variable Boolean function can also be uniquely represented as a multivariate polynomial over $\mathbb{F}_2$: $f(x) = \sum_{c\in\mathbb{F}_2^n} a_c x^c$, $a_c \in \mathbb{F}_2$, called algebraic normal form (ANF). The algebraic degree of f, denoted by $\deg(f)$, is defined as $\max\{\mathrm{wt}(c) | a_c \ne 0\}$.

The minimum distance between f and Boolean functions with degree at most r is called r-th order nonlinearity of f, denoted by $\mathrm{nl}_r(f)$. That is $\mathrm{nl}_r(f) = \min_{\deg(g)\le r} \mathrm{d}(f,g)$. It is called nonlinearity, denoted by $\mathrm{nl}(f)$, if $r = 1$.

3 Probabilistic Algebraic Attacks

Let f, g, h be n-variable Boolean functions, $g \ne 0$ and h of low degrees. Assume that f, g, h satisfy the scenario S4, i.e., $\Pr[gf = h] \approx 1$. Denote $A = \{x | g(x)f(x) = h(x)\}$, $B = \{x | h(x)f(x) = h(x)\}$. If $c \in A$, i.e., $g(c)f(c) = h(c)$, then $h(c)f(c) = g(c)f^2(c) = g(c)f(c) = h(c)$, and $x \in B$. Therefore, $A \subset B$. Then

$$\Pr[gf = h] = \Pr[x \in A] \le \Pr[x \in B] = \Pr[hf = h].$$

Hence

$$\max_{hf'=0} \Pr[f+1 = f'] \ge \Pr[h(f+1) = 0] = \Pr[hf = h] \ge \Pr[gf = h] \approx 1,$$

i.e., $f+1$ is equal to some Boolean function admitting h as an annihilator with probability ≈ 1.

Similarly to S4, if there is a Boolean function g of low degree such that $\Pr[gf = 0] \approx 1$, then probabilistic algebraic attacks are available. In this scenario, f is equal to some Boolean function admitting g as an annihilator with probability ≈ 1.

The two above scenarios can be reduced into one scenario that f is equal to some Boolean function of low algebraic immunity with probability ≈ 1. So, S4a and S4b can be concluded into the scenario:

S4′. There exists a nonzero function f' of low algebraic immunity such that $f = f'$ with probability $1-\varepsilon$.

3.1 Validity of Probabilistic Algebraic Attacks

Without loss of generality, we suppose that f coincides with S4b. Let $p = \Pr[g(x) = 0 | x \in 1_f]$ and $p_0 = \Pr[g(x) = 0]$. Now we consider the difference $\delta_f(g) = p - p_0$. If $\delta_f(g) \approx 0$, then solving the equation systems of $g(x) = 0 (x \in 1_f)$ is almost equivalent to solve the equation systems of $g(x) = 0$, and therefore it is unavailable to applying probabilistic algebraic attacks. Hence, probabilistic algebraic attacks make necessary that $\delta_f(g) \not\approx 0$. The value $\delta_f(g)$ reflects the validity of probabilistic algebraic attacks on f using the function g. The smaller $\delta_f(g)$ is, the worst probabilistic algebraic attacks behave; but not vice versa.

Theorem 1 *Let f be an n-variable balanced Boolean function. Then*

$$\max_{\deg(g)\le r} |\delta_f(g)| = \frac{2^{n-1} - \mathrm{nl}_r(f)}{2^n}.$$

If $\mathrm{nl}_r(f)$ is close to 2^{n-1}, then $\max|\delta_f(g)|$ is close to 0. And if $\mathrm{nl}_r(f)$ is close to 0, then $\max|\delta_f(g)|$ is close to $\frac{1}{2}$. This means that if the function f is balanced and of high r-th order nonlinearity, then f is also robust against r-th order probabilistic algebraic attacks to some extent.

4 Distance to Algebraic Immunity

In this section, we consider the set of n-variable Boolean functions with algebraic immunity $\le r$, and discuss the minimum distance between a Boolean function and that set.

Definition 2 *[3] The algebraic immunity of a function f, denoted by $\mathrm{AI}(f)$,is defined as*

$$\mathrm{AI}(f) = \min\{\deg(g) | gf = 0 \text{ or } g(f+1) = 0,\ g \ne 0\}.$$

Denote by $\mathcal{AI}_r = \{f \in \mathbf{B}_n | \mathrm{AI}(f) \le r\}$ the set of n-variable Boolean functions with algebraic immunity $\le r$. By convention $\mathcal{AI}_0 = \{0, 1\}$.

Proposition 1 *Let $r \geq 1$. Then $\mathcal{AI}_r = \{gh + c | g, h \in \mathbf{B}_n, c \in \mathbb{F}_2, 1 \leq \deg(g) \leq r\}$.*

It is significant to study the set $\mathcal{AI}_r$, since its complementary set $\mathbf{B}_n \backslash \mathcal{AI}_r$ contains all the functions with algebraic immunity $\geq r+1$. Then we introduce the notion of distance to algebraic immunity.

Definition 3 *The minimum distance between the Boolean function f and the set $\mathcal{AI}_r$ is called the distance to algebraic immunity r, denoted by $\mathrm{dai}_r(f)$, i.e.,*

$$\mathrm{dai}_r(f) = \mathrm{d}(f, \mathcal{AI}_r) = \min_{f' \in \mathcal{AI}_r} \mathrm{d}(f, f'). \tag{1}$$

Proposition 2 $\mathrm{dai}_r(f) = \min\limits_{1 \leq \deg(g) \leq r} \{\mathrm{d}(gf, 0), \mathrm{d}(gf, g)\}$.

The distance to algebraic immunity is similar to but not the same as the partial nonlinearity. For balanced f we have $\mathrm{nlp}_r(f) = 2\,\mathrm{dai}_r(f)$ while for the unbalanced case $\mathrm{nlp}_r(f)$ is always not an integer.

5 *k*-error Algebraic Immunity

Now, we consider the minimum algebraic immunity of the new functions after changing some values in the truth table of f. First, we introduce the notion of k-error algebraic immunity.

Definition 4 *Let $k \geq 0$ be an integer and $f \in \mathbf{B}_n$. The k-error algebraic immunity of the function f is defined as*

$$AI^k(f) = \min_{\mathrm{wt}(h) \leq k} \{\mathrm{AI}(f+h)\}.$$

It's well known that k-error algebraic immunity generalizes the notion of algebraic immunity. For small k, Boolean functions should have high k-error algebraic immunity.

By Definition 3 and Definition 4, we have the following result.

Corollary 1 *Let $f \in \mathbf{B}_n$. Then $\mathrm{dai}_r(f) = \min\{k | AI^k(f) \leq r\}$.*

By Corollary 1, any function f has $\mathrm{dai}_r(f)$-error algebraic immunity at most r, and has $(\mathrm{dai}_r(f) - 1)$-error algebraic immunity greater than r.

6 Bounds of Distance to Algebraic Immunity

Theorem 2 *Let $f \in \mathbf{B}_n$ and $\mathrm{wt}_{\min}(f) = \min\{\mathrm{wt}(f), \mathrm{wt}(f+1)\}$. Then*

$$dai_r(f) \leq \mathrm{wt}_{\min}(f) - \sum_{i=0}^{r} \binom{n}{i} + 1.$$

Corollary 2 *Let $n > 1$ be an odd integer and $f \in \mathbf{B}_n$. Then $dai_{\frac{n-1}{2}}(f) \leq 1$ and $AI^1(f) \leq \frac{n-1}{2}$.*

Braeken and Preneel [1] proved that $\mathrm{dai}_r(f) \leq 2^{n-r-1}$. In [4], Pometun observed that $dai_r(f) \leq \frac{1}{2}\,\mathrm{nl}_r(f)$ for balanced f.

Theorem 3 *Let $f \in \mathbf{B}_n$ and $\mathrm{wt}_{\max}(f) = \max\{\mathrm{wt}(f), \mathrm{wt}(f+1)\}$. Then*

$$\mathrm{dai}_r(f) \geq 2^{n-r-1} + \frac{1}{2}\,\mathrm{nl}_r(f) - \frac{1}{2}\,\mathrm{wt}_{\max}(f).$$

For large r, the bound of Theorem 3 may be negative. However, we only need consider small r in practice. By Theorem 3, we state that balanced functions is optimal among the functions of the same r-th order nonlinearity. This coincides with the viewpoint that balance is a very important property in cryptography for Boolean functions. For balanced f, if $\mathrm{nl}_r(f)$ is close to 2^{n-1}, then $\mathrm{dai}_r(f)$ is close to 2^{n-r-1}. This means that if the balanced function f has high r-th order nonlinearity, then f also has high $\mathrm{dai}_r(f)$. Again, it states that a balanced function of high r-th order nonlinearity can avoid the scenario S4$'$ to some extent.

From Theorem 3, we derive the lower bound of $\mathrm{dai}_r(f)$ for Boolean functions with a designated k-th order nonlinearity, and obtain some results as byproducts on the relations between k-th order nonlinearity and algebraic immunity.

Corollary 3 *Let $f \in \mathbf{B}_n$, $1 \leq r \leq k$. If $\mathrm{nl}_k(f) > \mathrm{wt}_{\max}(f) - 2^{n-k}$, then $\mathrm{dai}_r(f) > 2^{n-k-1}(2^{k-r} - 1)$ and $\mathrm{AI}(f) > k$.*

Corollary 4 *Let $f \in \mathbf{B}_n$. If $\mathrm{AI}(f) \leq k$, then $\mathrm{nl}_k(f) \leq \mathrm{wt}_{\max}(f) - 2^{n-k}$.*

Acknowledgement

The authors thank the anonymous referees for their valuable comments on this paper.

References

1. A. Braeken and B. Preneel. Probabilistic algebraic attacks. In 10th IMA international conference on cryptography and coding, 2005, number 3796 in Lecture Notes in Computer Science, pages 290–303. Springer-Verlag, 2005.
2. N. Courtois, W. Meier. Algebraic attacks on stream ciphers with linear feedback. Advances in Cryptology-EUROCRYPT 2003, LNCS 2656. Berlin, Heidelberg: Springer, 2003, 345–359.
3. W. Meier, E. Pasalic, C. Carlet. Algebraic attacks and decomposition of Boolean functions. Advances in Cryptology-EUROCRYPT 2004, LNCS 3027. Berlin, Heidelberg: Springer, 2004: 474–491.
4. S. Pometun. Study of Probabilistic Scenarios of Algebraic Attacks on Stream Ciphers. Journal of Automation and Information Sciences. 2009, 41(2): 67–80.

A Family of Weak Keys in HFE (and the Corresponding Practical Key-Recovery)

Charles Bouillaguet[1],
Pierre-Alain Fouque[1], Antoine Joux[2,3], and Joana Treger[2,4]

[1] Ecole Normale Supérieure
{charles.bouillaguet, pierre-alain.fouque}@ens.fr
[2] Université de Versailles-Saint Quentin
[3] DGA
antoine.joux@m4x.org
[4] ANSSI
joana.treger@ssi.gouv.fr

Abstract. The HFE (Hidden Field Equations) cryptosystem is one of the most interesting public-key multivariate scheme. It has been proposed more than 10 years ago by Patarin and seems to withstand the attacks that break many other multivariate schemes, since only subexponential ones have been proposed. The public key is a system of quadratic equations in many variables. These equations are generated from the composition of the secret elements: two linear mappings and a polynomial of small degree over an extension field. In this paper we show that there exist weak keys in HFE when the coefficients of the internal polynomial are defined in the ground field. In this case, we reduce the secret key recovery problem to an instance of the Isomorphism of Polynomials (IP) problem between the equations of the public key and themselves. Even though for schemes such as SFLASH or C^* the hardness of key-recovery relies on the hardness of the IP problem, this is normally not the case for HFE, since the internal polynomial is kept secret. However, when a weak key is used, we show how to recover all the components of the secret key in practical time, given a solution to an instance of the IP problem. This breaks in particular a variant of HFE proposed by Patarin to reduce the size of the public key and called the "subfield variant". Recovering the secret key takes a few minutes.
Key words: Cryptanalysis, multivariate cryptography, HFE, weak keys, Gröbner Bases.

1 Introduction

Multivariate cryptography is interesting from several points of view. First of all, it is based on a hard problem, namely solving system of multivariate equations, for which there only exist generic algorithms whose complexity is exponential in the worst case. Then, it has been proposed as an alternative to the RSA cryptosystem since there is no quantum algorithms for this problem. Finally, it is also appealing since the public operation does not require computations with large integers, and no crypto-processor is needed.

The HFE cryptosystem has been proposed in 1996 by Patarin in [28] in order to avoid his attack on the Matsumoto-Imai cryptosystem [23, 27]. This last scheme has also been called C^* and basically hides the power function $X \mapsto X^{1+q^\theta}$ in an extension field of degree n over $\mathbb{F}_q$, using two secret linear bijections S and T. In order to invert it, it suffices to remark that this power function, as the RSA power function, can be easily inverted provided $1 + q^\theta$ is invertible modulo $q^n - 1$. In [28], Patarin proposed to change the internal *known* monomial into a *secret* polynomial $\mathbf{f}$ of small degree. The legitimate user can still easily invert the public key since she knows S and T, and can invert the small degree polynomial using the Berlekamp algorithm for instance.

1.1 Related Works

From the adversary point of view, the action of S and T transforms the secret internal polynomial into a very sparse univariate polynomial of very high degree, as shown for instance by Kipnis and Shamir in [22].

A possible decryption attack would consist in inverting or factorizing this polynomial. However, there are no efficient algorithms to perform these tasks (an attempt can be found in [35]), and merely deciding the existence of roots is in fact NP-complete (*cf.* [22]).

HFE belongs to the category of public-key cryptosystems based on the hardness of computing a *functional decomposition*: given the composition of two functions f and g, can one identify the two components? Other examples include C^*, SFLASH [30], FAPKC [36], 2R [32] and McEliece [24]. With the exception of the latter, the former have all been broken because computing a functional decomposition was not as hard as expected. In the context of HFE, computing such a decomposition is related to decomposing the univariate representation of the public key, in order to recover the secret internal polynomial $\mathbf{f}$ as well as polynomial representations of S and T. Computing polynomial decompositions is a simple and natural mathematical problem which has a long history, going back to the works of Ritt and Ore in 1922 and 1930 respectively [34, 26]. Today, polynomial decomposition algorithms exist for some classes of polynomials over finite fields [37, 38], but no such algorithm is applicable to HFE. One step of the attack presented in this article amounts to computing a polynomial decomposition, and makes use of Gröbner bases.

The complexity of existing attacks, which all amount to solving systems of quadratic equations, depends on the degree d of the secret internal polynomial. When this degree is fixed, their complexity is polynomial in the security parameter n, although the exponent can be ridiculously large. In order for decryption to be polynomial, d must grow at most polynomially in n, and in that case the attacks are no longer polynomial. We consider this setting to be the most natural one to compare the asymptotic complexity of these attacks.

A simple decryption attack against HFE consists, given a ciphertext, in trying to solve the equations given by the public key. In 2003, Faugère and Joux experimentally showed that the HFE equations are not random systems of multivariate equations, because computing a Gröbner basis for these equations is much easier than the corresponding problem with random quadratic equations [16]. This allowed a custom implementation of the F5 algorithm [15] to break the first HFE challenge, for which the public key has 80 quadratic equations in 80 unknowns over $\mathbb{F}_2$. Later, Granboulan *et al.* [20] showed that specific algebraic properties of the HFE equations make the complexity of inverting HFE subexponential, in $\mathcal{O}\left(\exp\left(\log^2 n\right)\right)$.

In general, the hardness of recovering the secret key of HFE from the public key is unrelated to the Isomorphism of Polynomials (IP) problem [28], unless the internal polynomial is made public. A key recovery attack in the usual case where this polynomial is secret was presented in [22] and turns the problem of recovering T into an instance of the MinRank problem, the decisional version of which is NP-Complete [6]. Solving this instance of MinRank can be done by solving an overdetermined system of about n^2 quadratic equations in about $n \cdot \log d$ variables. The complexity of solving these equations is subexponential in $\mathcal{O}\left(\exp\left(\log^3 n\right)\right)$. This is too high to be practical, even for parameters corresponding to the HFE challenge that was broken.

These results show that HFE is not as robust as expected. However, can we consider HFE really broken? Is it still a viable alternative to RSA?

The cryptographic community often perceives HFE as broken, because of the practical attacks on some instances, and vastly lost both trust and interest in it. We would like to argue that the situation of HFE is slightly more complex. The complexity of some Gröbner basis algorithms, like F5 [15] is better understood [1] and allows to estimate the complexity of the decryption attacks, which remains relatively high for general instances. Moreover, standard modifications – such as removing some equations from the public key– destroy the algebraic structure presented by public key and that was exploited by Gröbner basis algorithms. HFE with Removed public equations is often called HFE^{-}, and suitable for a signature scheme. No attack faster than exhaustive search are known against HFE^{-}. In particular, the second HFE cryptanalytic challenge, with removed public equations, is currently far from being broken. Furthermore, it is suggested in [11] (based on experimentations) that the subexponential behavior of the Gröbner basis computation is mostly due to the fact that the computations are performed over $\mathbb{F}_2$, and that over odd-characteristic fields, computing a Gröbner basis of the public-key is no longer subexponential, but plainly exponential. This would mean that even when HFE is used for encryption, there are non-broken parameters.

All in all, HFE is comparatively in better shape than the SFLASH signature scheme for which polynomial time algorithms are known both to invert [13, 12] and to recover equivalent secret keys [18]. These attacks against SFLASH exploit the fact that multiplication matrices commute in some way with the internal monomial[1]. Then, it is possible to recover conjugates of the multiplications by the secret matrix S using simple linear algebra on the differential of the public key [18]. However, for general HFE, the multiplications no longer commute with the secret polynomial. Another issue is that we also need to recover the internal secret polynomial.

1.2 Our Results

In this paper, we consider the key recovery problem on a class of *weak keys* for HFE. As opposed to the decryption attack of Faugère and Joux [16], we recover an equivalent representation of the secret key that subsequently allows to inverse the trapdoor with the same complexity as the legitimate user. The weak instances we attack are defined by using an internal polynomial with coefficients in the ground field and not in the extension field as it was originally specified, or instances that are reducible to these specific ones (by considering equivalent transformations S and T, see section 3.1). Some instances belonging to this category were proposed by Patarin himself in [29] (an extended version of [28]) with the aim of reducing the size of the HFE public key (the so-called "subfield" variant). However, notice that the family of weak keys described here does not reduce to this subfield variant, and choosing the coefficients of the secret polynomial in the base field can seem rather natural. While in general, the hardness of the key-recovery does not depend on the hardness of the IP problem, we show that key recovery can be reduced to an instance of the IP problem, and that the solutions of this problem allow us to efficiently recover all the secret elements (or equivalent data). The latest IP algorithms allows to solve the instances in practice for realistic parameters set. To mount our attack, as in the SFLASH case [12], we try to find a commutation property to gain information about the secret key. In our attack, since multiplications no longer commute, we instead use the Frobenius map.

Coming back to the subfield variant, other schemes, including UOV [21] for instance, also have subfield variants, and the default in the design of an older version of SFLASH (v1) was to choose the secrets in a subfield. These schemes, or their subfield variants have all been broken: SFLASH v1 was attacked by Gilbert and Minier in [19], and subfield-UOV was shown to be insecure as well [4]. Although SFLASH and HFE share a similar structure, the Gilbert-Minier attack against SFLASH v1 cannot be applied to subfield-HFE, since it is based on Patarin's attack against C^{*}. Because this latter attack has no equivalent for HFE, there is no known attack against the subfield variant of HFE.

As mentioned above, the complexity of nearly all existing attacks depends on the degree of the internal secret polynomial. Even the most concrete and realistic threat, namely computing a Gröbner basis of the public-key, will become irrealistic if this degree is chosen high enough (a

[1] SFLASH is based on C^{*} and has a single internal monomial.

drawback is that decryption then becomes slower). A nice feature of the attack presented in this paper is that its asymptotic complexity is only marginally affected by the degree of the internal polynomial. As such, it be applied *in practice* to HFE instances on which existing attacks would be completely intractable. We also argue that under standard conjectures on the complexity of Gröbner basis computation, it is possible to establish that the complexity of our remains polynomial when the degree of the internal polynomial grows polynomially with n.

1.3 Organization of the Paper

Section 2 gathers some mathematical results, as well as basics on the HFE cryptosystem. In subsection 2.3, we give known results on the problem of finding isomorphisms of polynomials, that we need to mount our attack. Then, we describe our attack on the specific instances of HFE mentioned before in section 4. Finally, in section 5, to illustrate the attack, we show that we can break in practice a wide range of realistic parameters, including the ones proposed by Patarin for the "subfield" variant.

2 About HFE

2.1 Mathematical Background

Extension Fields and Vector Spaces. Let $\mathbb{K}$ be the finite field with q elements and $\mathbb{L}$ an extension of $\mathbb{K}$ of degree $n > 1$. $\mathbb{L}$ is isomorphic to $\mathbb{K}^n$ via an application φ. Hence, any application A defined over $\mathbb{L}$ can be seen as an application over $\mathbb{K}^n$ and conversely (just consider $\varphi^{-1} \circ A \circ \varphi$). Recall that any application over $\mathbb{L}$ is a polynomial of $\mathbb{L}[X]$.

The Frobenius Map. The application $\mathrm{F} : X \mapsto X^q$ over $\mathbb{L}$ is called the Frobenius map. It is an automorphism of $\mathbb{L}$ that fixes any element of $\mathbb{K}$. As a consequence, F can also be seen as a matrix $\mathrm{F} \in \mathrm{GL}_n(\mathbb{K})$. A polynomial $P \in \mathbb{L}[X]$ commutes with F if and only if its coefficients are in $\mathbb{K}$.

Linear Polynomials. Let M be an endomorphism of $\mathbb{K}^n$. It can be represented by a matrix over $\mathbb{K}^n$, but also as a polynomial over $\mathbb{L}$. Such $\mathbb{K}$-linear (or "additive") polynomials only have monomials of degree q^i, for $0 \le i \le n-1$. In the sequel, we will always identify a $n \times n$ matrix over $\mathbb{K}$ with its polynomial representation over $\mathbb{L}$. The set of matrices commuting with F over $\mathcal{M}_n(\mathbb{K})$ is the $\mathbb{K}$-vector space of dimension n generated by $\left(\mathrm{F}^0, \mathrm{F}, \ldots, \mathrm{F}^{n-1}\right)$. We will also need the following lemma:

Lemma 1. *Let $M \in \mathrm{GL}_n(\mathbb{K})$ be an invertible matrix. If its polynomial representation has coefficients over $\mathbb{K}$, then it is also the case for its inverse.*

Proof. If the polynomial representation of M has coefficients in $\mathbb{K}$, then M commutes with F. This implies that M^{-1} also commutes with F, which in turn implies that the polynomial representation of M^{-1} has coefficients in $\mathbb{K}$. □

2.2 Hidden Field Equations

The HFE scheme was designed in [28] by Patarin. Notice that specific variations of HFE do exist, but we will focus on the basic HFE scheme. Let us briefly recall its mechanism.

Let $\mathbb{K} = \mathbb{F}_q$ be the field with q elements. The HFE secret key is made up of an extension $\mathbb{L}$ of degree n over $\mathbb{K}$, a low-degree polynomial $\mathbf{f}$ over $\mathbb{L}$, and two invertible affine mappings S and T over $\mathbb{K}^n$. The secret polynomial $\mathbf{f}$ has the following particular shape:

$$\mathbf{f}(X) = \sum_{\substack{0 \le i,j \le n \\ q^i+q^j \le d}} a_{i,j} X^{q^i+q^j} + \sum_{\substack{0 \le k \le n \\ q^k \le d}} b_k X^{q^k} + c, \tag{1}$$

with the $a_{i,j}$, the b_k and c lying in $\mathbb{L}$. Polynomials with the same shape as $\mathbf{f}$ are called HFE polynomials[2]. Because decryption requires to invert $\mathbf{f}$, the maximum degree of $\mathbf{f}$, denoted by d, has to be chosen so that the factorization of $\mathbf{f}$ over $\mathbb{L}$ is efficient. All known algorithms for factorizing over finite fields are at least quadratic in the degree of the polynomial, which restricts d to values smaller than about 2^{16}. It also makes sense to consider degree bounds of the form $d = 2 \cdot q^D$, because in equation (1), we may then consider the sum over values of i and j smaller than D. Because the iterates of the Frobenius are $\mathbb{K}$-linear, then $\mathbf{f}$, seen as a transformation of $\mathbb{K}^n$, can be represented represented by a vector of n *quadratic* polynomials in n variables over $\mathbb{K}$. This property extends to the public key of the basic HFE scheme, defined by $\mathbf{PK} = T \circ \circ S$.

2.3 Known Algorithms for Finding Isomorphisms of Polynomials

In this section we briefly list the known techniques to solve the Isomorphism of Polynomials (IP) problem. This problem was first introduced in [28], and its hardness underlies for instance the hardness of the key-recovery of the C^* scheme. As already mentionned, the security of HFE does not rely in general on the hardness of this problem, but in the case of the attack on specific instances presented in this paper, we reduce the recovery of the private key to solving an instance of the IP problem, which happens to be tractable in some cases (e.g. the "subfield" case, see section 5).

Recall that finding a polynomial isomorphism between two vectors of multivariate polynomials $\mathbf{a}$ and $\mathbf{b}$ means finding two invertible matrices U and V in $\mathrm{GL}_n(\mathbb{F}_q)$, as well as two vectors c and d in ${\mathbb{F}_q}^n$ such that:

$$\mathbf{b}\left(x\right) = V\left(\mathbf{a}\left(U \cdot x + c\right)\right) + d \tag{2}$$

It has been proved that the IP problem is not NP-hard, unless the polynomial hierarchy collapses [17]. On the other hand, IP has been shown to be as hard as Graph-Isomorphism [33], for which no polynomial algorithms are known.

The first non-trivial algorithm for IP, known as the "To and Fro" technique, is due to Courtois *et al.* [33]. In its primitive form, this algorithm assumes the ability to inverse the polynomial systems, and has therefore an exponential complexity. A theoretical, birthday-based version of this algorithm is claimed to solve the problem in time and space $\mathcal{O}\left(q^{n/2}\right)$ if $c = d = 0$.

In [17], Faugère and Perret present a new technique for solving IP when $c = d = 0$. The idea is to model the problem as an algebraic system of equations and solve it by means of Gröbner bases [5, 8]. This technique has the advantage over the previous one that it is deterministic and always succeeds. On the down side, its complexity is hard to predict. In practice, it turns out to be efficient for instances of IP where the coefficients of all the monomials of all degree of $\mathbf{a}$ and $\mathbf{b}$ are randomly chosen in $\mathbb{F}_q$. For random instances of IP, the practical complexity of [17] has empirically been observed to be $\mathcal{O}\left(n^9\right)$.

[2] They were also studied much earlier in a completely different context by Dembowski and Ostrom [9], so they are sometimes referred to as D–O polynomials in the literature.

More recently, a faster algorithm dealing with the same class of instances ($c = d = 0$) provably achieves an expected complexity of $\mathcal{O}\left(n^6\right)$ on random instances [3]. This means that solving such random instances is feasible in practice for $n = 128$ or $n = 256$, which are the highest values encountered in practical HFE settings.

No polynomial algorithm is known when $c \neq 0$ or $d \neq 0$, or when $\mathbf{a}$ and $\mathbf{b}$ are homogeneous, and these are the most recurring settings in multivariate cryptography. However, it was also shown in [3] that it is possible to solve these hard instances without first guessing c and d. This enables a birthday-based algorithm to deal in practice with these hard instances in time $n^{3.5} \cdot q^{n/2}$.

3 A Specific Family of HFE Secret Polynomials

3.1 A commutation property for some HFE Secret Polynomials

To begin with, let us consider the *à la C^** case, where the secret polynomial $\mathbf{f}$ over $\mathbb{L}$ is just a monomial $a \cdot X^{q^i+q^j}$, $a \in \mathbb{L}$. Then the public key $\mathbf{PK} = T \circ \mathbf{f} \circ S$ can also be written as $T' \circ X^{q^i+q^j} \circ S$, by "absorbing" the multiplication by the constant a into T. As a consequence, without loss of generality, we can suppose that $a \in \mathbb{K}$ (or even that $a = 1$, but $a \in \mathbb{K}$ suffices for our purpose).

This secret monomial has some special commutation properties, which were used in [12, 13] to perform attacks on SFLASH. More precisely, composing it on the right hand size by multiplications M_x by an element x is equivalent to composing it on the left hand size by $M_{x^{q^i+q^j}}$. Another property, not used in [12, 13], is that it also commutes with the Frobenius map F and its iterates.

When we consider a more general HFE secret polynomial, the two commutation properties no longer hold. However, if we restrict the HFE polynomials to have their coefficients in $\mathbb{K}$, we lose the first property but the commutation with the Frobenius map still remains. Such instances can be represented by figure 1. Notice that if the coefficients of the HFE secret polynomial $\mathbf{f}$ can all be written as the product of the same element u of $\mathbb{L}$ with an element of $\mathbb{K}$, then by considering an equivalent transformation T made up of the original T and the multiplication by this factor u, we can suppose that $\mathbf{f}$ has coefficients in $\mathbb{K}$ too. This is the same as for the monomial case explained above. The same goes if we can modify the transformation S by composing it with a multiplication by an element m over $\mathbb{L}$, in such a way that the remaining polynomial $\mathbf{f}$ has coefficients in $\mathbb{K}$. In fact, this is all about equivalent secret keys [39]. Finally, the commutation property with the Frobenius can be exploited for instances of the type:

$$\mathbf{f}(X) = \sum_{\substack{0 \leq i,j \leq n \\ q^i+q^j \leq d}} u \cdot a_{i,j} \cdot m^{q^i+q^j} \cdot X^{q^i+q^j} + \sum_{\substack{0 \leq k \leq n \\ q^k \leq d}} u \cdot b_k \cdot m^{q^k} \cdot X^{q^k} + u \cdot c, \qquad (3)$$

with $a_{i,j}, b_k, c \in \mathbb{K}$, $u \in \mathbb{L}$, and $m \in \mathbb{L}$.

$$\begin{tikzcd}
\mathbb{K}^n \arrow[r, "S"] \arrow[rrr, bend left, "\mathbf{PK}"] & \mathbb{K}^n \arrow[r, dashed, "\mathbf{f}"] & \mathbb{K}^n \arrow[r, "T"] & \mathbb{K}^n
\end{tikzcd}$$

Fig. 1. $\mathbf{PK} = T \circ \mathbf{f} \circ S$. The broken arrow indicates that $\mathbf{f}$ has coefficients in $\mathbb{K}$.

Our key-recovery attack described in Section 4 exploits this second commutation property and could also apply to monomial instances of HFE, but this is not the point of this paper, as it has already been efficiently done [12, 13, 18]. Notice that legitimate users could easily check whether the internal polynomial of their secret keys belongs to the family verifying this commutation property, but we do not detail this fact here. In the next subsection, we give bounds on the number of HFE secret polynomials belonging to the family described.

3.2 An Estimation of the Cardinal of this Family

Let us study the cardinal of the family highlighted in section 3.1. Recall that we consider HFE polynomials with coefficients in $\mathbb{K}$, but also polynomials that can be written $M_\delta \circ \mathbf{f}' \circ M_\lambda$, where $\mathbf{f}'$ has coefficients in $\mathbb{K}$, M_δ (respectively M_λ) is the multiplication by $\delta \in \mathbb{L} \setminus \mathbb{K}$ (respectively $\lambda \in \mathbb{L} \setminus \mathbb{K}$).

Amongst this set of polynomials defined by $M_\delta \circ \mathbf{f}' \circ M_\lambda$, there are some instances for which $\mathbf{f}'$ commutes with multiplication applications. We already mentionned the case where $\mathbf{f}'$ is a monomial, but actually, such a commutative property may arise when $\mathbf{f}'$ is made up of two terms or sometimes more. Let us detail this point. We can write:

$$\begin{aligned} \mathbf{f}' \circ M_\lambda(X) &= \sum_{i,j} a_{i,j} \cdot (\lambda \cdot X)^{q^i+q^j} + \sum_i b_i \cdot (\lambda \cdot X)^{q^i} + c \\ &= \sum_{i,j} a_{i,j} \cdot \lambda^{q^i+q^j} \cdot X^{q^i+q^j} + \sum_i b_i \cdot \lambda^{q^i} \cdot X^{q^i} + c; \qquad (4) \\ M_\delta \circ \mathbf{f}'(X) &= \sum_{i,j} a_{i,j} \cdot \delta \cdot X^{q^i+q^j} + \sum_i b_i \cdot \delta \cdot X^{q^i} + \delta \cdot c. \qquad (5) \end{aligned}$$

When $\mathbb{K} = \mathbb{F}_2$ and $c = 0$, or $\mathbb{K} \neq \mathbb{F}_2$ and $\mathbf{f}'$ is homogeneous, we can sometimes have an equality between the two right-hand sides of equations (4) and (5) above. Let us consider these instances and suppose we have such an equality. As a result, we have some conditions on δ and λ. More precisely, we see that for a commutation property with multiplications to exist, the conditions, when consistents, often force λ and δ to live in a strict subfield of $\mathbb{L}$, which turns out to be most probably $\mathbb{K}$ as soon as $\mathbf{f}'$ has more than two terms. These are very specific instances, but have to be considered if we want to evaluate the number of HFE secret keys which belong to the family described in subsection 3.1. We have:

Proposition 2. *The number of HFE secret polynomials belonging to the family considered in this paper is lower-bounded by:*

i) $(q^{\frac{(D+1)(D+2)}{2}} - 1) \cdot q^{D+2} \cdot (q^n - q)$, when $\mathbb{K} \neq \mathbb{F}_2$,
ii) $(q^{\frac{D(D+1)}{2}} - 1) \cdot q^{D+3} \cdot (q^n - q)$, when $\mathbb{K} = \mathbb{F}_2$,

corresponding to $\mathcal{O}\left(q^{D^2+n}\right)$*, and upper-bounded by:*

i) $\left((q^{\frac{(D+1)(D+2)}{2}} - 1) \cdot q^{D+2} - \frac{(D+1)(D+2)}{2} \cdot (q-1)\right) \cdot (q^n - q)^2 + \frac{(D+1)(D+2)}{2} \cdot (q-1) \cdot (q^n - q)$, when $\mathbb{K} \neq \mathbb{F}_2$,
ii) $\left((q^{\frac{D(D+1)}{2}} - 1) \cdot q^{D+2} - \frac{(D+1)(D+2)}{2} \cdot (q-1)\right) \cdot (q^n - q)^2 + \frac{(D+1)(D+2)}{2} \cdot (q-1) \cdot (q^n - q)$, when $\mathbb{K} = \mathbb{F}_2$,

which corresponds to $\mathcal{O}\left(q^{D^2+2n}\right)$.

Proof. An HFE polynomial has $\frac{(D+1)(D+2)}{2} + (D+1) + 1 = \frac{D(D+5)}{2} + 3$ terms when $\mathbb{K} \neq \mathbb{F}_2$, $\frac{(D+1)(D+2)}{2} - (D+1) + (D+2) + 1 = \frac{(D+1)(D+2)}{2} + 2$ when $\mathbb{K} = \mathbb{F}_2$. We have:

1. $\mathbb{K}$ has q elements. The number of HFE polynomials with coefficients in $\mathbb{K} \neq \mathbb{F}_2$ is $q^{\frac{D(D+5)}{2}+3}$ ($q^{\frac{(D+1)(D+2)}{2}+2}$ when $\mathbb{K} = \mathbb{F}_2$). We however focus on polynomials over $\mathbb{K}$, which are non-linear over $\mathbb{K}$. This gives $(q^{\frac{(D+1)(D+2)}{2}} - 1) \cdot q^{D+2}$ polynomials when $\mathbb{K} \neq \mathbb{F}_2$, $(q^{\frac{(D+1)(D+2)}{2}-(D+1)} - 1) \cdot q^{D+3} = (q^{\frac{D(D+1)}{2}} - 1) \cdot q^{D+3}$ otherwise.
2. The number of elements belonging to $\mathbb{L} \backslash \mathbb{K}$ is $q^n - q$. Hence, the number of HFE polynomials that can be written as a polynomials with coefficients in $\mathbb{K}$ (a polynomial of point 1.), composed on the left by a mulitplication M_λ, for $\lambda \in \mathbb{L} \setminus \mathbb{K}$, is:

$$\begin{aligned} &(q^{\frac{(D+1)(D+2)}{2}} - 1) \cdot q^{D+2} \cdot (q^n - q), && \text{when } \mathbb{K} \neq \mathbb{F}_2, \\ &(q^{\frac{D(D+1)}{2}} - 1) \cdot q^{D+3} \cdot (q^n - q), && \text{when } \mathbb{K} = \mathbb{F}_2. \end{aligned}$$

This corresponds to the lower bound of the proposition.

Now, to evaluate the exact number of polynomials belonging to our family, we should evaluate the number of polynomials that can be written as in point 2, composed on the right by a multiplication by an element of $\mathbb{L} \backslash \mathbb{K}$. However, we saw that some polynomials have the property that composing them by a multiplication on the left is equivalent to composing them by another multiplication on the right. We thus have to be carefull not to count such poynomial twice. Amongst these polynomials, only monomials have this property for sure. The number of $\mathbb{K}$-quadratic monomials over $\mathbb{L}$ with coefficients in $\mathbb{K}$ is $\frac{(D+1)(D+2)}{2} \cdot (q-1)$ or $\frac{(D(D+1)}{2}) \cdot (q-1)$, whether $\mathbb{K} = \mathbb{F}_2$ or not. This allows to establish the upper-bound of the proposition:

$$\begin{aligned} &\left((q^{\frac{(D+1)(D+2)}{2}} - 1) \cdot q^{D+2} - \frac{(D+1)(D+2)}{2} \cdot (q-1) \right) \cdot (q^n - q)^2 \\ &+ \frac{(D+1)(D+2)}{2} \cdot (q-1) \cdot (q^n - q), && \text{when } \mathbb{K} \neq \mathbb{F}_2, \\ &\left((q^{\frac{D(D+1)}{2}} - 1) \cdot q^{D+2} - \frac{(D+1)(D+2)}{2} \cdot (q-1) \right) \cdot (q^n - q)^2 \\ &+ \frac{(D+1)(D+2)}{2} \cdot (q-1) \cdot (q^n - q), && \text{when } \mathbb{K} = \mathbb{F}_2. \end{aligned}$$

□

We show in this paper that for all HFE secret polynomials with coefficients in $\mathbb{K}$ (or more precisely, the family described by equation (3) and number in proposition 2), the security in fact relies on the hardness of the IP problem. Moreover, in the cases where this IP problem can be solved (see subsection 2.3), then we can also recover an efficient secret key (maybe different from the original one) in practical time.

4 The Attack

The attack being quite complex, let us give an overview. A pseudo-code of the attack is given in fig. 2. First, we show that the representation of $\mathbb{L}$ can be supposed public. Then, as already mentioned in Section 3.1, we use the commutation of the Frobenius map with the secret polynomials considered, which propagates to the public key **PK**. This key property allows us to recover applications closely related to S and T. An interpolation of **PK** combined with these applications then gives us a polynomial over $\mathbb{K}$ from which we recover $\mathbf{f}$ or an equivalent low-degree polynomial by computing a functional decomposition. In any case, we obtain the original secret key or a different one that allows us to decrypt as efficiently as the secret key owner. All these assertions are detailed and justified in this section.

Fig. 2 Pseudo-code of the attack

```
Require: An HFE public key $\mathbf{PK}$, generated by $(T, \mathbf{f}, S)$ such that $\mathbf{f} \in \mathbb{K}[X]$.
Ensure: An equivalent secret key: $(T', \mathbf{f}', S')$, with $\deg \mathbf{f}' \leq \deg \mathbf{f}$.
 1: // section 4.2
 2: repeat
 3:    Let $(U, V)$ be a (random) solution to the IP problem: $U \circ \mathbf{PK} = \mathbf{PK} \circ V$.
 4: until $U$ and F are similar
 5: // section 4.3
 6: for all $i_0$ in $[1; n-1]$ prime with $n$ do
 7:    Let $k = {i_0}^{-1} \mod n$.
 8:    Compute $\widetilde{S}, \widetilde{T}$ such that $\mathrm{F} = \widetilde{S} \circ V^k \circ \widetilde{S}^{-1} = \widetilde{T}^{-1} \circ U^k \circ \widetilde{T}$.
 9:    // section 4.4
10:    Interpolate $\mathbf{g} = \widetilde{T}^{-1} \circ \mathbf{PK} \circ \widetilde{S}^{-1}$.
11:    if $\mathbf{g}$ has coefficients in $\mathbb{K}$ then
12:       // section 4.5
13:       Compute $F_1, F_2$ and $\mathbf{f}_2$, such that $\mathbf{g} \circ F_1 = F_2^{-1} \circ \mathbf{f}_2$.
14:       return $\left(\widetilde{T} \cdot F_2^{-1}, \mathbf{f}_2, F_1^{-1} \cdot \widetilde{S}\right)$
15:    end if
16: end for
```

4.1 Equivalent Secret Keys : Irrelevance of Hiding the Extension

It is shown in [39] that there are many equivalent keys in HFE. As a consequence, one can assume S and T to be *linear* bijections (as opposed to affine), and arbitrarily choose their value on one point. Indeed, it is possible to compensate changes in S and T by changes in $\mathbf{f}$. In the restricted setting where $\mathbf{f}$ is assumed to have coefficients in the base field $\mathbb{K}$ (instead of the extension field $\mathbb{L}$), this is no longer possible, because there is not enough freedom if we want to keep $\mathbf{f} \in \mathbb{K}[X]$.

However, what holds in general and still holds for our family of secret keys is that the assumption of keeping the representation φ of $\mathbb{L}$ secret is not necessary. This was already mentioned in the original paper [28], and as a matter of fact, the specifications of both Quartz [31] and SFLASH make the description of the extension public. In any case, it is possible to generate the *same* public key from the *same* secret polynomial, while fixing an arbitrary correspondence between $\mathbb{L}$ and $\mathbb{K}^n$. It simply requires slight modifications on S and T:

Proposition 3. *Let* $\mathbf{SK} = (T, \mathbf{f}, S, \varphi)$ *be an HFE secret key. Then for any choice of an isomorphism* φ' *between* $\mathbb{L}$ *and* $\mathbb{K}^n$*, there exist two affine bijections* S' *and* T' *such that* $\mathbf{SK}' = (T', \mathbf{f}, S', \varphi')$ *is equivalent to* $\mathbf{SK}$ *(i.e., generates the same public key).*

Proof. If φ' denotes another isomorphism between $\mathbb{L}$ and $\mathbb{K}^n$, then $\phi = \varphi' \circ \varphi^{-1}$ is a $\mathbb{K}$-linear invertible application such that $\varphi' = \phi \circ \varphi$. Using the correspondence φ', the composition $T' \circ \mathbf{f} \circ S'$ is also equal to $\mathbf{PK}$, where $T' = \phi \circ T$ and $S' = S \circ \phi^{-1}$. □

Thus, the assumption of keeping $\mathbb{L}$ secret does not have an influence on the security of HFE. Would the extension be secret, one could just arbitrarily fix its own and be guaranteed that an equivalent secret key exists. As a consequence, throughout the sequel, we assume that the description of $\mathbb{L}$ is public.

4.2 A Useful Property of HFE Secret Polynomials Lying in $\mathbb{K}[X]$

Recall from Section 2.1 that because $\mathbf{f}$ has coefficients in $\mathbb{K}$, then it commutes with F:

$$\mathbf{f} \circ \mathrm{F}(X) = \mathrm{F} \circ \mathbf{f}(X) \tag{6}$$

Patarin left as an open problem whether this property has security implications or not. We shall demonstrate that it does indeed. Most importantly, this property is detectable on the public-key.

Proposition 4. *There exists non-trivial polynomial isomorphisms between the public key and itself. More precisely, the invertible mapping ψ defined below transforms a matrix M that commutes with $\mathbf{f}$ into a solution of the polynomial automorphism of the public-key:*

$$\psi : M \mapsto \left(T \cdot M^{-1} \cdot T^{-1}, S^{-1} \cdot M \cdot S\right)$$

As a consequence, $\psi(F), \ldots, \psi\left(F^{n-1}\right)$ are non-trivial isomorphisms between $\mathbf{PK}$ and itself.

Proof. Let M be a matrix such that $\mathbf{f} \circ M = M \circ \mathbf{f}$. Then we get:

$$\begin{aligned} \mathbf{PK} \circ (S^{-1} \circ M \circ S) &= T \circ \mathbf{f} \circ S \circ S^{-1} \circ M \circ S \\ &= T \circ M \circ \mathbf{f} \circ S \\ &= (T \circ M \circ T^{-1}) \circ \mathbf{PK} \\ \Leftrightarrow \mathbf{PK} &= (T \circ M \circ T^{-1})^{-1} \circ \mathbf{PK} \circ (S^{-1} \circ M \circ S) \end{aligned}$$

Then, because of (6), $\psi(F), \ldots, \psi\left(F^{n-1}\right)$ are automorphisms of the public key. □

Remark 1. The existence of other solutions besides those mentioned in proposition 4 is extremely unlikely. Indeed, this would imply the existence of other linear applications commuting with the (non-linear) internal polynomial. However, besides the monomial instances, where multiplication matrices commute in some sense with $\mathbf{f}$, we are not aware of instances that would verify such a property. Thus, if we consider a particular solution of the problem of retrieving an automorphism of the public-key, we can assume that it is $\psi\left(F^{i_0}\right)$, for some unknown power i_0.

Hardness of the IP Problem. We discussed algorithms for solving the IP problem in subsection 2.3. In our setting, the conditions for which the polynomial-time IP algorithms are applicable are:

i) The secret transformations S and T are linear (as opposed to affine).
ii) The b_k coefficients of (1) are not all zero.
iii) The c coefficient of (1) is non-zero.

The first condition can only be satisfied if choosing linear S and T was a deliberate decision (otherwise it will only happen with negligible probability). There are good reasons of doing so: first it reduces a bit the size of the private key. Second, in general, because of the existence of equivalent keys, it can be assumed that S and T are linear. However, we emphasize that this last fact is *no longer true* if the internal polynomial $\mathbf{f}$ is chosen in $\mathbb{K}[X]$ instead of $\mathbb{L}[X]$ (section 4.1).

A sequence of bad design decisions could still lead to the combination of a restricted $\mathbf{f}$ *and* linear S and T.

The second condition will always be satisfied with high probability, and the third will be satisfied with probability $1/q$. It must be noted that if $c = 0$ in (1), then the public-key sends zero to zero, which might not be desirable.

In the case where S and T are affine, the situation is much more painful, and breaking the IP instance in practice requires a workload of $q'^{n/2}$ if the coefficients of S live in $\mathbb{F}_{q'}$. In the case of the "subfield variant" though, since $q' = 2$ and n is small enough, breaking the IP instances is still tractable (see section 5).

4.3 Retrieving "nearly S" and "nearly T" Applications

Let us assume that we have found an automorphism $(U, V) = \psi\left(\mathrm{F}^{i_0}\right)$ of the public-key, for some unknown integer i_0 in the interval $[1; n-1]$. The whole point of the attack is to "extract" enough information about S and T from this automorphism. For this purpose, the value of i_0 has to be known, and it is required that i_0 and n be relatively prime. This latter condition can be easily checked for: F^i and F^j are similar if and only if $\gcd(i,n) = \gcd(j,n)$. Therefore, i_0 is relatively prime with n if U and F are similar. If it turns out not to be the case, we take an other automorphism of $\mathbf{PK}$, until it passes the test. Since there are $\phi(n)$ values of i_0 that are prime with n, we expect to check $n/\phi(n) = \mathcal{O}(\log\log n)$ candidates.

To find out the actual value of i_0, we simply guess its value, and check whether the remaining steps of the attack are carried out successfully. Fortunately, there is a way to discard bad guesses systematically before the most computationally expensive step of the attack, as we will explain in section 4.4.

With the preceding notations, we have the following result:

Proposition 5. *Let* $(U, V) = \psi\left(\mathrm{F}^{i_0}\right)$*, with* $\gcd(i_0, n) = 1$*. Let* k *be such that* $k \cdot i_0 = 1 \bmod n$*.*

i) There exist $\widetilde{S}$*,* $\widetilde{T}$ *in* $\mathrm{GL}_n(\mathbb{K})$ *such that* $\mathrm{F} = \widetilde{S} \circ V^k \circ \widetilde{S}^{-1}$ *and* $\mathrm{F} = \widetilde{T}^{-1} \circ U^k \circ \widetilde{T}$*.*

ii) Both $\widetilde{S} \cdot S^{-1}$ *and* $\widetilde{T} \cdot T^{-1}$ *commute with* F*.*

Proof. *i)* We know that U and V are both similar to F^{i_0}. Thus U^k and V^k are both similar to $\mathrm{F}^{i_0 \cdot k} = \mathrm{F}^{1 \bmod n} = \mathrm{F}$.

ii) Let us consider the case of $\widetilde{S}$ (something similar holds for $\widetilde{T}$). We have:

$$\begin{aligned}
\mathrm{F} &= \widetilde{S} \circ V^k \circ \widetilde{S}^{-1} \\
&= \widetilde{S} \circ S^{-1} \circ \mathrm{F}^{i_0 \cdot k} \circ S \circ \widetilde{S}^{-1} \\
&= \widetilde{S} \circ S^{-1} \circ \mathrm{F} \circ S \circ \widetilde{S}^{-1}
\end{aligned}$$

And thus $\mathrm{F} \circ \widetilde{S} \circ S^{-1} = \widetilde{S} \circ S^{-1} \circ \mathrm{F}$. □

In practice, $\widetilde{S}$ and $\widetilde{T}$ can be found very efficiently through linear algebra, given that i_0 is known. Note that for now, this proposition cannot be used to test whether our current guess for i_0 is correct, since we do not know S.

4.4 Building an Equivalent Secret Key

The information about S (resp. T) contained in $\widetilde{S}$ (resp. $\widetilde{T}$) can be used to cancel the action of S and T on the public key. It follows from proposition 5 (see also section 2) that $F_1 = \widetilde{S} \cdot S^{-1}$ and $F_2 = T^{-1} \cdot \widetilde{T}$ are linear combinations over $\mathbb{K}$ of powers of F. We immediately obtain that:

$$\begin{aligned}
\widetilde{T}^{-1} \circ \mathbf{PK} \circ \widetilde{S}^{-1} &= F_2^{-1} \circ T^{-1} \circ T \circ \mathbf{f} \circ S \circ S^{-1} \circ F_1^{-1} \\
&= F_2^{-1} \circ \mathbf{f} \circ F_1^{-1}. \qquad (7)
\end{aligned}$$

11

We therefore define:

$$\mathbf{g} = \widetilde{T}^{-1} \circ \mathbf{PK} \circ \widetilde{S}^{-1} \bmod \left(X^{q^n} - X\right)$$

Because the HFE polynomials are stable by left and right composition by additive polynomials and by reduction modulo $X^{q^n} - X$, the "peeled off" polynomial $\mathbf{g}$ is still an HFE polynomial. Thus $\mathbf{g}$ has $\mathcal{O}\left(n^2\right)$ coefficients, and that they can be uniquely determined in polynomial time by interpolation (this was noted in [22]. Note that there would not be a unique solution if we did not perform the modular reduction of $\mathbf{g}$). By doing so, we obtain an equivalent secret key, namely $\left(\widetilde{T}, \mathbf{g}, \widetilde{S}\right)$.

By itself, this equivalent key is not particularly useful, since the degree of $\mathbf{g}$ is typically q^n, and we are therefore still facing our initial task of factorizing a sparse polynomial of very high degree. However, $\mathbf{g}$ has a very important property which brings us one step closer to the original secret-key:

Proposition 6. *The coefficients of* $\mathbf{g}$ *are in* $\mathbb{K}$ *(and not in* $\mathbb{L}$*).*

Proof. By hypothesis, the coefficients of $\mathbf{f}$ are in $\mathbb{K}$. From proposition 5, we have that the coefficients of the polynomial representation of F_1 and F_2 are in $\mathbb{K}$, then, so are those of the polynomial representations of ${F_1}^{-1}$ and ${F_2}^{-1}$ (by lemma 1). $\square$

The result of proposition 6 is illustrated in figure 3. This figure also helps remembering how the applications introduced so far intervene.

$$\begin{tikzcd}
\mathbb{K}^n \arrow[r, "S"] \arrow[rrr, bend left, "\mathbf{PK}"] \arrow[dr, "\widetilde{S}"'] & \mathbb{K}^n \arrow[r, dashed, "\mathbf{f}"] \arrow[d, dashed, "F_1"'] & \mathbb{K}^n \arrow[r, "T"] & \mathbb{K}^n \\
 & \mathbb{K}^n \arrow[r, dashed, "\mathbf{g}"] & \mathbb{K}^n \arrow[u, dashed, "F_2"'] \arrow[ur, "\widetilde{T}"'] &
\end{tikzcd}$$

Fig. 3: $\mathbf{PK} = T \circ \mathbf{f} \circ S = \widetilde{T} \circ \mathbf{g} \circ \widetilde{S}$. Broken arrows stand for applications with coefficients in $\mathbb{K}$.

This proposition can be used to verify if our guess for i_0 was right. Indeed, if $\mathbf{g}$ is found not to be in $\mathbb{K}[X]$, then the guess was wrong. We are aware that the fact that $\mathbf{g} \in \mathbb{K}[X]$ does not rigorously prove that we have found the right value of i_0. However, it does not matter, as $\mathbf{g} \in \mathbb{K}[X]$ is sufficient for the subsequent step to work.

4.5 Recovering a Low-Degree Equivalent Secret Key

To be useful, an equivalent secret key must have an internal polynomial of low degree. We now show how to obtain one, by actually computing the decomposition given by equation (7) of Section 4.4. This is in fact a *much easier* problem than computing the equivalent decomposition on the original public key, because we deal with applications whose coefficients belong to $\mathbb{K}$. They are then left invariant by the Frobenius (hence by F_1 and F_2), which implies that the problem of finding the decomposition reduces to finding a solution of an overdefined system of quadratic equations. This system can be solved in practical time by computing a Gröbner basis, as we now show. To this end, we introduce the following notations:

$$F_1(X)=\sum_{k=0}^{n-1}x_kX^{q^k}\qquad F_1^{-1}(X)=\sum_{k=0}^{n-1}y_kX^{q^k}$$
$$F_2(X)=\sum_{k=0}^{n-1}z_kX^{q^k}\qquad F_2^{-1}(X)=\sum_{k=0}^{n-1}t_kX^{q^k}$$

$$\begin{aligned}\mathbf{g}(X)&=\sum_{q^i+q^j<q^n}a_{ij}X^{q^i+q^j}+\sum_{i=0}^{n-1}b_iX^{q^i}+c\\ \mathbf{f}_2(X)&=\sum_{q^i+q^j\leq d}e_{ij}X^{q^i+q^j}+\sum_{q^i\leq d}f_iX^{q^i}+g\end{aligned}$$

Then, we consider the following polynomial equation, also represented by figue 4.5, obtained by composing both sides of equation (7) of Section 4.4 with F_1:

$$\mathbf{g}\circ F_1=F_2^{-1}\circ\mathbf{f}_2. \tag{8}$$

$$\begin{tikzcd}\mathbb{K}^n \arrow[r, dotted, "\mathbf{f}"] \arrow[d, dotted, "F_1"'] & \mathbb{K}^n \\ \mathbb{K}^n \arrow[r, "\mathbf{g}"] & \mathbb{K}^n \arrow[u, dotted, "F_2"']\end{tikzcd}$$

Fig. 4: $\mathbf{g}=F_2^{-1}\circ\mathbf{f}\circ F_1^{-1}$. Broken arrows stand for applications with unknown coefficients.

The left-hand side becomes:

$$\begin{aligned}\mathbf{g}\circ F_1&=\sum a_{ij}\left(\sum_{k=0}^{n-1}x_kX^{q^k}\right)^{q^i+q^j}+\sum b_i\left(\sum_{k=0}^{n-1}x_kX^{q^k}\right)^{q^i}+c\\ &=\sum_{i,j,k,l}a_{ij}\cdot x_k\cdot x_l\cdot X^{q^{i+k}+q^{l+j}}+\sum_{i,k}b_i\cdot x_k\cdot X^{q^{i+k}}+c\end{aligned}$$

We obtain a polynomial whose coefficients are quadratic in the coefficients of F_1. Now, let us compute the right-hand side:

$$\begin{aligned}F_2^{-1}\circ\mathbf{f}_2&=\sum_{k=0}^{n-1}t_k\left(\sum_{q^i+q^j\leq d}e_{ij}X^{q^i+q^j}+\sum_{q^i\leq d}f_iX^{q^i}+g\right)^{q^k}\\ &=\sum_{i,j,k}t_k\cdot e_{ij}\cdot X^{q^{i+k}+q^{j+k}}+\sum_{i,k}t_k\cdot f_i\cdot X^{q^{i+k}}+g\cdot\sum_k t_k\end{aligned}$$

We again obtain a polynomial whose coefficients are quadratic in the coefficients of both $\mathbf{f}_2$ and F_2^{-1}. Reducing both sides modulo $X^{q^n}-X$ and identifying coefficient-wise the two sides of equation (8) yields a system of $\mathcal{O}\left(n^2\right)$ quadratic equations in $\mathcal{O}\left(n + D^2\right)$ unknowns. However, these equations admit many parasitic solutions (for example, $F_1 = \mathbf{f}_2 = 0$). To avoid these, we also encode the fact that F_1 and F_2 are invertible. We describe how we encode the invertibility of F_1, as this is similar for F_2. We start from the equation: $F_1 \circ F_1^{-1}(X) = X$: because the coefficients of the left-hand side are quadratic in the x_k's and y_k's, we obtain n quadratic equations by reducing the LHS modulo $X^{q^n} - X$ and equating the coefficients on both sides of the equation. Note that this also introduce in all $2n$ additional unknowns (n for the coefficients of F_1^{-1} and n for the coefficients of F_2).

All in all, assuming that the degree of $\mathbf{f}$ is $d = 2q^D$, this yields $n(n+3)/2 + 1$ equations in $4n + D(D+5)/2 + 4$ variables, not counting eventual field equations (one per variable). The existence of at least one solution is guaranteed, because of equation (7) of Section 4.4, as long as we picked the right power of the Frobenius matrix in section 4.2. In fact, even though we just need one, we know that many solutions exist: for instance because the Frobenius commutes with everything in equation (8), we can take a particular solution, compose both F_2^{-1} and F_1 with the Frobenius, and obtain a new solution.

It turns out that these equations can be solved efficiently, even though the number of variables is higher than what is usually tractable, because it is very overdetermined: we have $\mathcal{O}\left(n^2\right)$ equations in $\mathcal{O}\left(n + D^2\right)$ variables, and D has to be small for decryption to be efficient (*i.e.*, $D = \mathcal{O}(\log n)$). In this setting, computing a Gröbner basis turns out to be feasible in practice.

Conjecture 1. The Gröbner basis of a system of random quadratic equations with the same number of variable and polynomials as our equations can be computed by manipulating polynomials of degree at most 8. Thus, it can be computed in time at most $\mathcal{O}\left(n^{24}\right)$ by the F_4 or F_5 algorithms [14, 15]. This is true if D is fixed, or even if grows polynomially with $\log n$.

Justification of the Conjecture. We argue that the complexity of computing a Gröbner basis of our equations is fact polynomial under realistic assumptions, although in the general case the algorithms involved in the computation are simply or doubly exponential.

The usual strategy to solve such an overdefined system of equations is to compute a Gröbner basis for the graded reverse lexicographic order, since it is easier, and then to convert it to a Gröbner basis for the lexicographic order. Let us recall that the complexity of all known Gröbner bases algorithms depends on the *degree of regularity* of the system [7, 1]. This corresponds to the maximal degree of polynomials manipulated during a Gröbner basis computation. If d_{reg} is the degree of regularity of an ideal $I \subset k[x_1, \ldots, x_m]$, then the complexity of computing a Gröbner basis of I using the F_5 algorithm [15] is upper-bounded by:

$$\mathcal{O}\left(\binom{n + d_{reg}}{d_{reg}}^{\omega}\right) = \mathcal{O}\left(n^{\omega \cdot d_{reg}}\right)$$

where ω is the linear algebra constant (between 2 and 3). In general, it is a difficult problem to know *a priori* the degree of regularity, although lower-bounds were shown in the context of the analysis of the XL algorithm [10].

To upper-bound the complexity of our Gröbner-basis computation, we use an existing approximation of the degree of regularity that applies to *regular* and *semi-regular* system of equations (*i.e.*, in which the equations are "as independent as possible". For a formal definition, see [1]). It is conjectured that the proportion of semi-regular systems goes to 1 when n goes to $+\infty$. Therefore, we will assume that for large n a random system is almost surely semi-regular (which is to some extent a worst-case assumption, as it usually means that our system is not easier to solve than the others). The coefficients of the Hilbert series associated with the ideal generated by a semi-regular sequence coincide with those of the series expansion of the function

$f(z) = \left(1 - z^2\right)^m / (1-z)^n$, up to the degree of regularity. The degree of regularity is the smallest degree d such that the coefficient of degree d in the series expansion of $f(z)$ is not strictly positive. This property enables an explicit computation of the degree of regularity for given values of m and n.

Furthermore, Bardet *et al.* [1] give asymptotic developments of the expression of the degree of regularity in the case of $\alpha \cdot n$ equations in n variables, where α is a constant greater than 1. While this result is not directly applicable to our case (because we have about αn^2 equations), we use it to derive a heuristic expression of the degree of regularity for systems of $\alpha \cdot n^2$.

When there are $\alpha \cdot n$ semi-regular quadratic equations in n variables, [1] gives:

$$d_{reg} = n\left(\alpha - \frac{1}{2} - \sqrt{\alpha(\alpha-1)}\right) - \frac{a_1}{2(\alpha(\alpha-1))^{\frac{1}{6}}} n^{\frac{1}{3}} - \left(2 - \frac{2\alpha - 1}{4\sqrt{\alpha(\alpha-1)}}\right) + \mathcal{O}\left(1/n^{1/3}\right),$$

$$\text{with } a_1 \approx -2.33811. \quad (9)$$

While we are well-aware that it is not theoretically justified (because equation (9) is established for a constant α), we now set $\alpha = \beta n$, and express d_{reg} as a function of β. This yields

$$d_{reg} = \frac{1}{8\beta} - \frac{a_1}{2\beta^{1/3}} - \frac{3}{2} + \mathcal{O}\left(1/n\right). \quad (10)$$

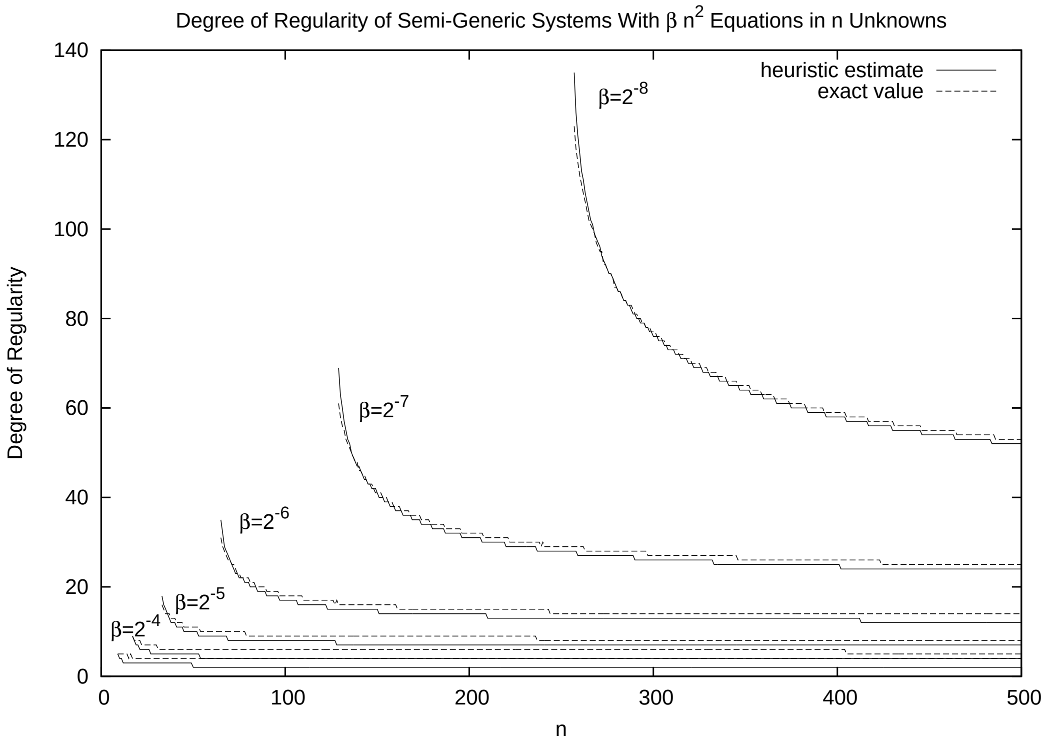
This heuristic result can be empirically checked to be rather precise, for various values of β and n, as shown in fig. 5(a). When n grows to infinity, it seems that the degree of regularity converges to a constant, an approximation of which is given by (10). We now apply this result to our setting:

1. Consider that D is fixed. Then when n becomes big, we have $\beta = 1/32$. Equation (10) then yields $d_{reg} = 7$ for large n (actually computing it using the Hilbert series gives a value of 8 for big n). Computing the Gröbner basis can thus be achieved with complexity $\mathcal{O}\left(n^{8\omega}\right)$.
2. Consider that the degree of $\mathbf{f}$ grows polynomially with n, which means that $D = \mathcal{O}\left(\log n\right)$. In that case we have $\beta = 1/32 + \mathcal{O}\left(\frac{\log n}{n}\right)$, and equation (10) still yields $d_{reg} \approx 7$ for large n.

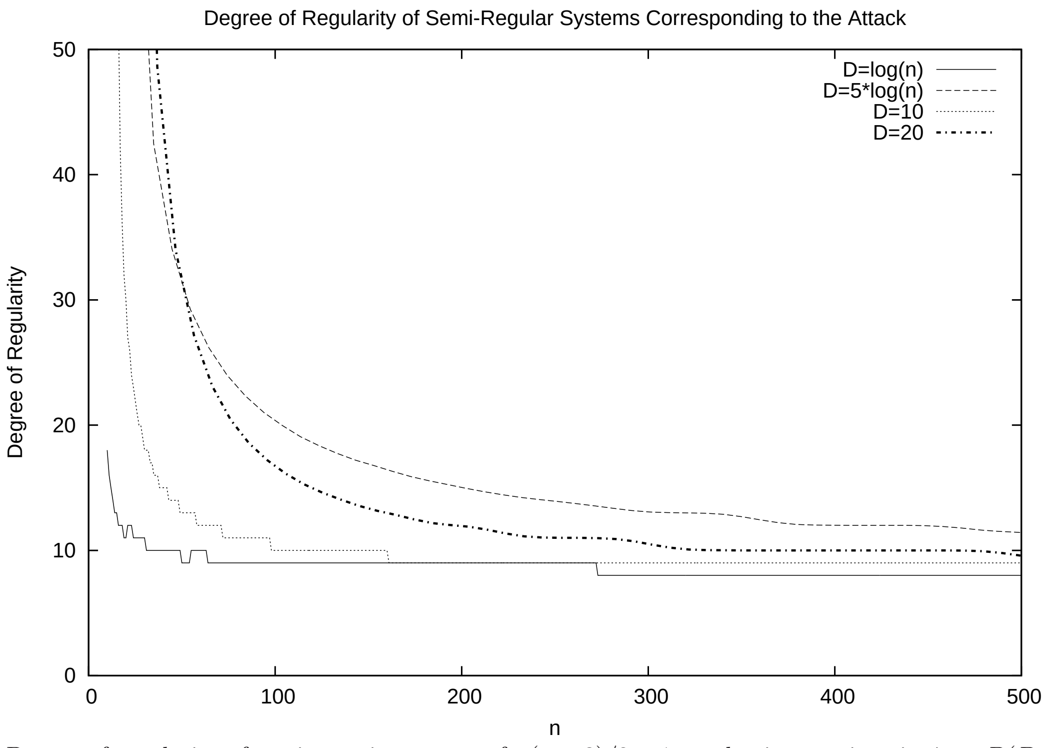
This shows that even in the more general setting the computation of the Gröbner basis should be polynomial, and the degree of the polynomials should not increase beyond a given threshold. Fig. 5(b) shows the degree of regularity of systems having the same parameters as those considered in the attack.

Comments and Practical Results. While the result conjectured above means that computing the polynomial decomposition we are dealing with should be polynomial, some remarks are in order. First, our equations are not random, not to mention semi-regular. This follows from the fact that they admit many solution, while a random overdetermined system has no solutions with overwhelming probability. Next, our experiments (for various values of n and D) indicate that a Gröbner basis can be computed by manipulating polynomials of degree at most 3, leading to an empirical complexity of $\mathcal{O}\left(n^9\right)$. Our equations are thus *easier* to solve than random systems with the same parameters.

Once the equations are solved, we recover an equivalent secret-key $\left(\widetilde{T} \cdot F_2^{-1}, \mathbf{f}_2, F_1^{-1} \cdot \widetilde{S}\right)$, which allows us to decrypt with the same time complexity as the legitimate user, since $\mathbf{f}_2$ has essentially the same degree as $\mathbf{f}$.



(a) Comparison between the heuristic estimate and the actual values of the degree of regularity for $\alpha \cdot n^2$ quadratic equations in n unknowns.



(b) Degree of regularity of semi-generic systems of $n(n+3)/2+1$ quadratic equations in $4n+D(D+5)/2+4$ variables.

16

5 Applications and Experiments

We programmed the HFE key-generation and encryption, as well as the attack, in the MAGMA [2] computer-algebra system. We do not claim that our implementation is efficient, nor reflects what kind of performances can be obtained in encryption. All the experiments were run on one core of an Intel 2.3Ghz Xeon "Nehalem" computer with 74 Gbyte of RAM. We tested our attack on several sets of parameters described below. We forged the solution of the IP instance from the knowledge of the secret S and T. The actual timings are given in figure 5.

Weak Keys. We first tested the attack on realistically-sized weak keys, corresponding to parameters set A,B and C. The chosen parameters allows the encryption or signature of 256, 134 and 97 bits respectively. We choose the degree of the internal polynomial very conservatively (*i.e.*, much higher than what was proposed for the HFE challenges, and high enough to make decryption painfully slow). To make the IP part of the attack feasible, we choose the secret bijections S and T to be linear (as opposed to affine). Then solving the IP instance is a matter of seconds with the techniques presented in [3]. We emphasize that none of the existing attack can be close to practical on parameter sets A and B.

Patarin's "Subfield" Variant of HFE. In order to reduce the size of the public key, Patarin suggested in [29] a "subfield" variant of HFE, in which the coefficients of the quadratic equations of **PK** live in a subfield $\Bbbk$ of $\mathbb{K}$. If $\mathbb{K} = \mathbb{F}_{256}$ and $k = \mathbb{F}_2$, this reduces the size of the public key by a factor of 8. To achieve this, the coefficients of S and T, the coefficients of the defining polynomial of the extension field $\mathbb{L}$, and the coefficients of the internal polynomial $\mathbf{f}$ have to be chosen in $\Bbbk$ (instead of $\mathbb{K}$ or $\mathbb{L}$ for the latter). S and T will be affine, so the polynomial-time IP algorithms do not apply in this case.

In order for the reduction of the public key size to be effective, $\mathbb{K}$ has to be relatively big and $\Bbbk$ relatively small. The former implies that D cannot be very huge, otherwise decryption is impractical, while the latter means little entropy in the internal polynomial. This opens a possible way of attack, consisting in guessing $\mathbf{f}$ and then solving the IP problem to recover S and T. We shall compare the attack presented in this paper with this simple one.

Patarin's "concrete proposal" is parameter set D in fig. 5. For practical decryption, we have to choose $D = 2$ (yielding an internal polynomial of degree at most 131072), and decryption can take at most 4 minutes on our machine. The internal polynomial has at most 10 terms with coefficients in $\mathbb{F}_2$. The simple "guess-$\mathbf{f}$-then-IP" key recovery attack therefore needs to solve 2^{10} affine IP instances for which $q = 2$ and $n = 29$. Such instances are in fact tractable even with older techniques (though no one ever noticed it), for instance using the "to-and-fro" algorithm of [33]. In that case, the "guess-then-IP" attack has a workload of 2^{68}. With the new attack presented in this paper, and the more advanced IP techniques described in [3], solving the IP instance takes about one second, and our attack takes less than one minute.

To show that the "subfield" variant is broken beyond repair, we show that it is possible to attack in practice parameters twice as big as the concrete proposal. This is parameter set E. The internal polynomial now has 21 terms, so the simple attack requires breaking 2^{21} affine instances of the IP problem with $q = 2$ and $n = 59$. According to [3], breaking one of these instance should take about one month using inexpensive hardware, with a workload of about 2^{59}. The "guess-then-IP" attack is here clearly impractical with a complexity of 2^{80}. Our attack requires one month to break the IP instance, plus about 4 hours for the remaining steps.

Parameter set	A	B	C	D	E
block size (bits)	256	134	97	232	236
q	256	4	2	256	16
N	32	67	97	29	59
$\deg \mathbf{f}$	131072	131072	128	131072	131072
coefficients of $\mathbf{f}$ in	$\mathbb{F}_{256}$	$\mathbb{F}_4$	$\mathbb{F}_2$	$\mathbb{F}_2$	$\mathbb{F}_2$
S and T	linear	linear	linear	affine	affine
coefficients of S, T in	$\mathbb{F}_{256}$	$\mathbb{F}_4$	$\mathbb{F}_2$	$\mathbb{F}_2$	$\mathbb{F}_2$
Terms in $\mathbf{f}$	10	54	29	10	21
size of **PK** (bits)	143'616	314'364	461'138	13'485	107'970
IP	polynomial			$\approx$ 1s	$\approx$ 5 weeks
Interpolation of $\mathbf{g}$ (once)	79s	30 min	140 min	51s	23min
Gröbner	7h	1 day	1 week	45s	3h
Variables / Equations	136 / 593	322/4947	423/10028	124 / 494	253 / 1889
Memory required	2.1Gbyte	45Gbyte	180Gbyte	350Mbyte	13.9Gbyte
Order Change	15s	30 min	4h	0s	30s

Fig. 5: Timings for the Attack

6 Conclusion

In this paper, we considered a special family of HFE instances, where the internal secret polynomial is defined over the base field $\mathbb{K}$ instead of the extension field $\mathbb{L}$. We show that, in that case, there are non-trivial isomorphisms of polynomials between the corresponding public key and itself. Interestingly, finding such an isomorphism suffices to completely recover (in practical time) a secret-key that allows fast decryption.

References

1. Bardet, M., Faugère, J.C., Salvy, B., Yang, B.Y.: Asymptotic Behaviour of the Degree of Regularity of Semi-Regular Polynomial Systems. In: MEGA'05. (2005) Eighth International Symposium on Effective Methods in Algebraic Geometry, Porto Conte, Alghero, Sardinia (Italy), May 27th – June 1st.
2. Bosma, W., Cannon, J.J., Playoust, C.: The Magma Algebra System I: The User Language. J. Symb. Comput. **24**(3/4) (1997) 235–265
3. Bouillaguet, C., Faugère, J.C., Fouque, P.A., Pérret, L.: Isomorphism of Polynomials : New Results (October 2010) unpublished manuscript. Available at: `http://www.di.ens.fr/~bouillaguet/pub/ip.pdf`.
4. Braeken, A., Wolf, C., Preneel, B.: A Study of the Security of Unbalanced Oil and Vinegar Signature Schemes. In Menezes, A., ed.: CT-RSA. Volume 3376 of Lecture Notes in Computer Science., Springer (2005) 29–43
5. Buchberger, B.: Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal. PhD thesis, University of Innsbruck (1965)
6. Buss, J.F., Frandsen, G.S., Shallit, J.: The Computational Complexity of Some Problems of Linear Algebra. J. Comput. Syst. Sci. **58**(3) (1999) 572–596
7. Cox, D.A., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra, 3/e (Undergraduate Texts in Mathematics). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2007)
8. Cox, D.A., Little, J.B., O'Shea, D.: Ideals, Varieties and Algorithms. Springer (2005)
9. Dembowski, P., Ostrom, T.G.: Planes of Order n with Collineation Groups of Order n^2. Mathematische Zeitschrift **103**(3) (1968) 239–258

10. Diem, C.: The xl-algorithm and a conjecture from commutative algebra. In Lee, P.J., ed.: ASIACRYPT. Volume 3329 of Lecture Notes in Computer Science., Springer (2004) 323–337
11. Ding, J., Schmidt, D., Werner, F.: Algebraic Attack on HFE Revisited. In Wu, T.C., Lei, C.L., Rijmen, V., Lee, D.T., eds.: ISC. Volume 5222 of Lecture Notes in Computer Science., Springer (2008) 215–227
12. Dubois, V., Fouque, P.A., Shamir, A., Stern, J.: Practical Cryptanalysis of SFLASH. In: CRYPTO. Volume 4622., Springer (2007) 1–12
13. Dubois, V., Fouque, P.A., Stern, J.: Cryptanalysis of SFLASH with Slightly Modified Parameters. In: EUROCRYPT. Volume 4515., Springer (2007) 264–275
14. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases (F4). Journal of Pure and Applied Algebra **139**(1-3) (June 1999) 61–88
15. Faugère, J.C.: A New Efficient Algorithm for Computing Gröbner Bases Without Reduction to Zero (F5). In: ISSAC '02: Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, New York, NY, USA, ACM (2002) 75–83
16. Faugère, J.C., Joux, A.: Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases. In Boneh, D., ed.: CRYPTO. Volume 2729 of Lecture Notes in Computer Science., Springer (2003) 44–60
17. Faugère, J.C., Perret, L.: Polynomial Equivalence Problems: Algorithmic and Theoretical Aspects. In Vaudenay, S., ed.: EUROCRYPT. Volume 4004 of Lecture Notes in Computer Science., Springer (2006) 30–47
18. Fouque, P.A., Macario-Rat, G., Stern, J.: Key Recovery on Hidden Monomial Multivariate Schemes. In Smart, N.P., ed.: EUROCRYPT. Volume 4965 of Lecture Notes in Computer Science., Springer (2008) 19–30
19. Gilbert, H., Minier, M.: Cryptanalysis of SFLASH. In Knudsen, L.R., ed.: EUROCRYPT. Volume 2332 of Lecture Notes in Computer Science., Springer (2002) 288–298
20. Granboulan, L., Joux, A., Stern, J.: Inverting HFE Is Quasipolynomial. In Dwork, C., ed.: CRYPTO. Volume 4117 of Lecture Notes in Computer Science., Springer (2006) 345–356
21. Kipnis, A., Patarin, J., Goubin, L.: Unbalanced Oil and Vinegar Signature Schemes. In: EUROCRYPT. (1999) 206–222
22. Kipnis, A., Shamir, A.: Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization. In Wiener, M.J., ed.: CRYPTO. Volume 1666 of Lecture Notes in Computer Science., Springer (1999) 19–30
23. Matsumoto, T., Imai, H.: Public Quadratic Polynominal-Tuples for Efficient Signature-Verification and Message-Encryption. In: EUROCRYPT. (1988) 419–453
24. McEliece, R.: A Public-Key Cryptosystem Based on Algebraic Coding Theory (1978) DSN Progress Report 42-44.
25. Naccache, D., ed.: Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings. In Naccache, D., ed.: CT-RSA. Volume 2020 of Lecture Notes in Computer Science., Springer (2001)
26. Ore, O.: Contributions to The Theory of Finite Fields. Transactions A. M. S. **36** (1934) 243–274
27. Patarin, J.: Cryptoanalysis of the Matsumoto and Imai Public Key Scheme of Eurocrypt'88. In Coppersmith, D., ed.: CRYPTO. Volume 963 of Lecture Notes in Computer Science., Springer (1995) 248–261
28. Patarin, J.: Hidden fields equations (hfe) and isomorphisms of polynomials (ip): Two new families of asymmetric algorithms. In: EUROCRYPT. (1996) 33–48
29. Patarin, J.: Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms. In: EUROCRYPT. (1996) 33–48 Etended version available on `http://www.minrank.org/hfe.pdf`.
30. Patarin, J., Courtois, N., Goubin, L.: Flash, a fast multivariate signature algorithm. [25] 298–307
31. Patarin, J., Courtois, N., Goubin, L.: QUARTZ, 128-Bit Long Digital Signatures. [25] 282–297
32. Patarin, J., Goubin, L.: Asymmetric cryptography with s-boxes. In Han, Y., Okamoto, T., Qing, S., eds.: ICICS. Volume 1334 of Lecture Notes in Computer Science., Springer (1997) 369–380
33. Patarin, J., Goubin, L., Courtois, N.: Improved Algorithms for Isomorphisms of Polynomials. In: EUROCRYPT. (1998) 184–200
34. Ritt, J.F.: Prime and Composite Polynomials. American M. S. Trans. **23** (1922) 51–66

35. Sidorenko, A.V., Gabidulin, E.M.: The Weak Keys For HFE. In: 7th International Symposium on Communication Theory and Applications. (2003) 239–244
36. Tao, R.J., Chen, S.H.: Two varieties of finite automaton public key cryptosystem and digital signatures. Journal of computer science and technology **1**(1) (1986) 9–18
37. von zur Gathen, J.: Functional Decomposition of Polynomials: The Tame Case. J. Symb. Comput. **9**(3) (1990) 281–299
38. von zur Gathen, J.: Functional Decomposition of Polynomials: The Wild Case. J. Symb. Comput. **10**(5) (1990) 437–452
39. Wolf, C., Preneel, B.: Large Superfluous Keys in Multivariate Quadratic Asymmetric Systems. In Vaudenay, S., ed.: Public Key Cryptography. Volume 3386 of Lecture Notes in Computer Science., Springer (2005) 275–287

A Multivariate Signature Scheme with a Partially Cyclic Public Key

Albrecht Petzoldt[1], Stanislav Bulygin[2], and Johannes Buchmann[1,2]

[1] Technische Universität Darmstadt, Department of Computer Science
Hochschulstraße 10, 64289 Darmstadt, Germany
{apetzoldt,buchmann}@cdc.informatik.tu-darmstadt.de

[2] Center for Advanced Security Research Darmstadt - CASED
Mornewegstraße 32, 64293 Darmstadt, Germany
{johannes.buchmann,Stanislav.Bulygin}@cased.de

Abstract. Multivariate public key cryptography is one of the main approaches to guarantee the security of communication in the post-quantum world. Due to its high efficiency and modest computational requirements, multivariate cryptography seems especially appropriate for signature schemes on low cost devices. However, multivariate schemes are not yet much used, mainly because of the large size of the public key. In this paper we present a new idea to reduce the public key size of multivariate cryptosystems by proposing a multivariate signature scheme with a partially cyclic public key. The scheme is based on the UOV-Scheme of Kipnis and Patarin, but reduces the size of the public key by up to 83 %.

Keywords: Multivariate Cryptography, UOV Signature Scheme, reduced key size

1 Introduction

When quantum computers arrive, cryptosystems based on numbertheoretic problems such as factoring or discrete logarithms will become insecure. So, to guarantee the security of communication in the post-quantum world, alternatives to classical public key schemes are needed. Besides lattice-, code and hash-based cryptosystems, Multivariate public key cryptography [BB08], [DG06] is one of the main approaches to achieve this goal. Since it requires only modest computational resources, multivariate schemes seem to be appropriate for the use on low cost devices. However, these schemes are not yet widely used, mainly because the large size of their public and private keys.

The basic idea behind multivariate cryptography is to choose a system $\mathcal{Q}$ of m quadratic polynomials in n variables which can be easily inverted (central map). After that one chooses two affine invertible maps $\mathcal{S}$ and $\mathcal{T}$ to hide the structure of the central map. The public key of the cryptosystem is the composed map $\mathcal{P} = \mathcal{S} \circ \mathcal{Q} \circ \mathcal{T}$ which is difficult to invert. The private key consists of $\mathcal{S}$, $\mathcal{Q}$ and $\mathcal{T}$ and therefore allows to invert $\mathcal{P}$.

In the last years, a lot of work was done to find ways how to reduce the key size of multivariate schemes. One way to achieve this is by choosing the coefficients of the private maps out of smaller fields (e.g. $GF(16)$ instead of $GF(256)$). However, this increases the signature length [CC08]. Another way to reduce the size of the private key is by using sparse central polynomials, which is done for example in the TTS schemes of Yang and Chen [YC05]. While these attempts mainly look for ways to reduce the size of the private key, we are concentrating on the public key. In this paper we present a new idea to reduce the public key size of multivariate schemes. The main idea behind our scheme is to choose the coefficients of the private key in such a way that the corresponding public key gets a compact structure.

As we find, by this strategy it is not possible to create a scheme with a completely cyclic key, but we can achieve that a major part of the public key will be cyclic. So, we will have $M_P = (B|C)$, where B is a partially circulant matrix obtained from a vector $\mathbf{b}$ and C is a matrix with no apparent structure. Thus we have to store only the vector $\mathbf{b}$ and the matrix C, which reduces the size of the public key by up to 83 %. Furthermore, the number of multiplications needed in the verification process can be reduced by 41 %.

The rest of the paper is organized as follows:
In Section 2 we describe the Oil and Vinegar (OV) Signature Scheme, which is the basis of our new scheme and look at certain properties of its public key. A detailed description of our new scheme can be found in Section 3 . In Section 4 we give a security analysis of our scheme. Section 5 gives example parameters and compares the scheme to other multivariate schemes of the UOV family, and Section 6 concludes the paper with future research questions.

2 The (Unbalanced) Oil and Vinegar Signature Scheme

One way to create easily invertible multivariate quadratic systems is the principle of Oil and Vinegar, which was first proposed by J. Patarin in [Pa97].

Let K be a finite field (e.g. $K = GF(2^8)$). Let o and v be two integers and set $n = o + v$. Patarin suggested to choose $o = v$. After this original scheme was broken by Kipnis and Shamir in [KS98], it was recommended in [KP99] to choose $v > o$ (Unbalanced Oil and Vinegar (UOV)). In this section we describe the more general approach UOV.
We set $V = \{1, \dots, v\}$ and $O = \{v + 1, \dots, n\}$. Of the n variables $x_1, \dots, x_n$ we call $x_1, \dots, x_v$ the Vinegar variables and $x_{v+1}, \dots, x_n$ Oil variables. We define o quadratic polynomials $q_k(\mathbf{x}) = q_k(x_1, \dots, x_n)$ by

$$q_k(\mathbf{x}) = \sum_{i\in V,\ j\in O} \alpha_{ij}^{(k)} x_i x_j + \sum_{i,j\in V,\ i\le j} \beta_{ij}^{(k)} x_i x_j + \sum_{i\in V\cup O} \gamma_i^{(k)} x_i + \eta^{(k)} \ (1 \le k \le o)$$

Note that Oil and Vinegar variables are not fully mixed, just like oil and vinegar in a salad dressing.

The map $\mathcal{Q} = (q_1(\mathbf{x}), \dots, q_o(\mathbf{x}))$ can be easily inverted. First, we choose the values of the v Vinegar variables $x_1, \dots, x_v$ at random. Therefore we get a system of o linear equations in the o variables $x_{v+1}, \dots, x_n$ which can be solved by Gaussian Elimination. (If the system does not have a solution, choose other values of $x_1, \dots, x_v$ and try again).

To hide the structure of $\mathcal{Q}$ in the public key one concatenates it with a linear invertible map $\mathcal{T}$. So, the public key of the UOV scheme is given as

$$\mathcal{P} = \mathcal{Q} \circ \mathcal{T} \tag{1}$$

Remark 1: In opposite to other multivariate schemes the second affine map $\mathcal{S}$ is not needed for the security of UOV. So it can be dropped.

2.1 Properties of the public key

The public key $\mathcal{P}$ of the UOV scheme consitsts of o quadratic polynomials in n variables.

$$\begin{aligned} P &= (p^{(1)}, \dots, p^{(o)}) \\ &= \left(\sum_{i=1}^{n} \sum_{j=i}^{n} p_{ij}^{(1)} x_i x_j + \sum_{i=1}^{n} p_i^{(1)} x_i + p_0^{(1)}, \dots, \sum_{i=1}^{n} \sum_{j=i}^{n} p_{ij}^{(o)} x_i x_j + \sum_{i=1}^{n} p_i^{(o)} x_i + p_0^{(o)} \right) \end{aligned} \tag{2}$$

After having chosen a monomial ordering (e.g. graded lexicographical ordering), we can write down the public coefficients into an $o \times \frac{(n+1)\cdot(n+2)}{2}$-matrix M_P.

$$M_P = \begin{pmatrix} p_{11}^{(1)} \; p_{12}^{(1)} \; \dots \; p_{nn}^{(1)} \; p_1^{(1)} \; \dots \; p_n^{(1)} \; p_0^{(1)} \\ \vdots \qquad\qquad\qquad\qquad\qquad \vdots \\ p_{11}^{(o)} \; p_{12}^{(o)} \; \dots \; p_{nn}^{(o)} \; p_1^{(o)} \; \dots \; p_n^{(o)} \; p_0^{(o)} \end{pmatrix} = \begin{pmatrix} \pi_{11} & \dots & \pi_{1d} \\ \vdots & & \vdots \\ \pi_{m1} & \dots & \pi_{md} \end{pmatrix}, \tag{3}$$

where $d = \frac{(n+1)\cdot(n+2)}{2}$ is the number of columns in M_P. For the UOV scheme, the public key is given as

$$\mathcal{P} = \mathcal{Q} \circ \mathcal{T},$$

with a linear invertible map $\mathcal{T}$ and the central map $\mathcal{Q}$ (as defined in the previous subsection).

Due to equation (1), we get the following equations for the coefficients of the quadratic terms of the public key:

$$p_{ij}^{(r)} = \sum_{k=1}^{n} \sum_{l=k}^{n} \alpha_{kl}^{ij} \cdot q_{kl}^{(r)} = \sum_{k=1}^{v} \sum_{l=k}^{n} \alpha_{kl}^{ij} \cdot q_{kl}^{(r)} \; (1 \leq i \leq j \leq n, \; r = 1, \dots, o) \tag{4}$$

with

$$\alpha_{kl}^{ij} = \begin{cases} t_{ki} \cdot t_{li} & (i = j) \\ t_{ki} \cdot t_{lj} + t_{kj} \cdot t_{li} & \text{otherwise} \end{cases} \tag{5}$$

Note that the right hand side of equation (4) only contains coefficients of the quadratic terms of $\mathcal{Q}$ and coefficients of $\mathcal{T}$ and is linear in the former ones. The second "=" in equation (4) is due to the fact, that all the q_{ij} $(i, j \in O)$ are zero.

3 The Scheme

3.1 Construction

We denote $s := \frac{v\cdot(v+1)}{2} + o \cdot v$. For $i = 1, \dots, o$ we define two s-vectors $v_P^{(i)} = (p_{kl}^{(i)} | 1 \leq k \leq v, \; k \leq l \leq n)$ and $v_Q^{(i)} = (q_{kl}^{(i)} | \; 1 \leq k \leq v, \; k \leq l \leq n)$ containing the first quadratic coefficients of the public and private polynomials with respect to the graded lexicographical ordering.

Note that the vector $v_Q^{(i)}$ contains all the nonzero quadratic coefficients of the i-th central polynomial.

Additionally, we define an $s \times s$ matrix A containing the coefficients of the equations (4):

$$A = \left(\alpha_{kl}^{ij}\right) \; (1 \leq k \leq v, k \leq l \leq n \text{ for the rows, } \; 1 \leq i \leq v, i \leq j \leq n \; \text{ for the columns}), \text{i.e.}$$

$$A = \begin{pmatrix} \alpha_{11}^{11} & \alpha_{11}^{12} & \dots & \alpha_{11}^{vn} \\ \alpha_{12}^{11} & \alpha_{12}^{12} & \dots & \alpha_{12}^{vn} \\ \vdots & & & \vdots \\ \alpha_{vn}^{11} & \alpha_{vn}^{12} & \dots & \alpha_{vn}^{vn} \end{pmatrix}. \tag{6}$$

Therefore we have (for each $i = 1, \dots o$):

$$v_P^{(i)} = v_Q^{(i)} \cdot A. \tag{7}$$

We can use the equations (7) to create a UOV-like scheme with much smaller public key.

To build our new scheme, we assign the coefficients of $\mathcal{T}$ some random elements of K. Therefore, the entries of the matrix A can be computed by equation (5).

Thus equation (7) yields a linear relation between the vectors $v_P^{(i)}$ and $v_Q^{(i)}$.

To use this relation properly, we need the matrix A to be invertible. Assuming A being invertible, we can prove the following theorem:

Theorem: For every $\ell \leq \frac{v\cdot(v+1)}{2} + o \cdot v$, $\mathbf{b} = (b_1, \dots, b_\ell) \in_R K^\ell$ and invertible affine map $\mathcal{T} = (M_T, c_T) : K^n \to K^n$ there exist two quadratic maps $\mathcal{P}, \mathcal{Q} : K^n \to K^o$ such that

1. $\mathcal{Q}$ is a UOV map
2. we have $\mathcal{P} = \mathcal{Q} \circ \mathcal{T}$ as a composition of mappings and the entries of the matrix M_P fulfill

$$\pi_{ij} = b_{(j-i \mod \ell)+1} \quad (1 \leq i \leq o, 1 \leq j \leq \ell) \tag{8}$$

Remark 2: To justify the assumption of A being invertible, we carried out a number of experiments. For different values of o and v we created 1000 matrices A each time and tested, how many of them were invertible. Table 1 shows the results. As the table shows, the condition of A

(2^8,o,v)	(2,4)	(5,10)	(10,20)	(15,30)	(20,40)
% invertible	99.3	99.6	99.7	99.5	99.4

Table 1. Percentage of the matrices A being invertible

being invertible is nearly always complied.

Remark 3: When choosing $\ell < \frac{v\cdot(v+1)}{2} + o \cdot v$, one has to define the vectors $v_P^{(i)}$ and $v_Q^{(i)}$ and the matrix A in a slightly different way. We leave out the details here.

3.2 Description of the Scheme

Key Generation

1. For $\ell \leq \frac{v\cdot(v+1)}{2} + o \cdot v$ choose a vector $\mathbf{b} = (b_1, \ldots, b_\ell) \in_R K^\ell$.
2. Choose a linear map $\mathcal{T} = (M_T, c_T)$ at random. If M_T is not invertible, choose again.
3. Compute for T the corresponding matrix A (using equations (4) and (5)). If A is not invertible, go back to step 2.
4. For $i = 1, \ldots, o$ set
$$v_P^{(i)} = \mathcal{S}^{i-1}(\mathbf{b})$$
where $\mathcal{S}^i(\mathbf{b})$ is the circular right shift of the vector $\mathbf{b}$ by i positions.
5. Solve for $i = 1, \ldots, o$ the linear systems given by equation (7) to get the vectors $v_Q^{(i)}$ and therewith the quadratic coefficients of the central polynomials.
6. Choose the coefficients of the linear terms of the private polynomials at random.
7. Compute the public key as $\mathcal{P} = \mathcal{Q} \circ \mathcal{T}$.

The *public key* of the scheme consists of the vector $\mathbf{b}$ and the last $\frac{(n+1)\cdot(n+2)}{2} - \ell$ columns of M_P. The *private key* consists of the maps $\mathcal{Q}$ and $\mathcal{T}$.

Signature generation and verification The signature generation and verification works as in the case of the UOV Scheme. To *sign* a message with a hash value $h \in K^o$, one computes recursively $y = \mathcal{Q}^{-1}(h)$ and $z = \mathcal{T}^{-1}(y)$. The signature of the message is $z \in K^n$. (Here $\mathcal{Q}^{-1}(h)$ means finding one preimage of $h \in K^o$ under $\mathcal{Q}$, which we get by choosing the vinegar variables at random and solving the linear system for the oil variables.)
To *verify* the signature, one computes $w = \mathcal{P}(z)$. If $w = h$ holds, the signature is accepted, otherwise rejected.
The *size of the public key* is $\ell + o \cdot \left(\frac{(n+1)\cdot(n+2)}{2} - \ell\right) = o \cdot \frac{(n+1)\cdot(n+2)}{2} - (o-1)\cdot \ell$ field elements, the *size of the private key* is $o \cdot \left(\frac{v\cdot(v+1)}{2} + o \cdot v + n + 1\right) + n \cdot (n+1)$ field elements.

4 Security

In this part of the paper we analyse the security of our scheme. To do this, we consider the effects of the special structure of our public key under known attacks.

Direct attacks [BB08]: When attacking our scheme directly, one has to solve an underdetermined system of quadratic equations. So, before applying an algorithm like XL or a Groebner basis algorithm, one has to guess at at least v of the variables. We analysed the effects of this guessing on our public key and found, that a major part of the cyclic structure gets lost. The following experiments show, that the F_4-algorithm (as implemented in MAGMA) is not able to use the remaining structure in the key and can not solve the system faster than a UOV scheme with the same parameters.

$(2^8, o, v)$	(11,22)	(12,24)	(13,26)	(14,28)
UOV	6.2 m	0.9 h	6.8 h	47.1 h
cyclicUOV	6.1 m	0.9 h	6.8 h	47.0 h

Table 2. Direct attacks with MAGMA'S F_4-algorithm

UOV-Reconciliation [BB08]: In this attack one tries to find an equivalent private key by a basis change of the variables. To find such a change of basis, one has to solve a several overdetermined systems of multivariate quadratic equations which can be derived easily from the public key. We found that during the algebraic part of this attack much of the cyclic structure of the public key gets lost such that the resulting systems are very similar to those we looked at for direct attacks. Our experiments showed, that MAGMA can not solve the resulting systems faster than those obtained from a UOV scheme.

$(2^8, o, v)$	(10,20)	(11,22)	(12,24)	(13, 26)	(14,28)
UOV	55 s	390 s	3142 s	24316 s	173524 s
cyclicUOV	54 s	388 s	3144 s	24298 s	173352 s

Table 3. Running time of the Reconciliation attack

Rank attacks [GC00]: In this paragraph we compare the behavior of Rank attacks against the standard UOV and our scheme. For different values of o and v we created 100 instances of both schemes and computed the matrices P_i $(i = 1, \dots, o)$ representing the homogenous quadratic parts of the public polynomials. For every instance we formed 100 linear combinations H of the matrices P_i. We found that for both schemes nearly all of them were full rank and none had rank less than $n - 2$ (see Table 4). So, it seems that Rank attacks are hardly applicable.

UOV Attack [KP99]: The goal of this attack is to find the preimage of the oil subspace $\mathcal{O} = \{x \in K^n : x_1 = \dots = x_v = 0\}$ under the affine invertible transformation $\mathcal{T}$. To achieve this, one forms random linear combinations of the matrices P_i, multiplies them with the inverse of one of the P_i and looks for invertible subspaces of these matrices. For each pair (o, v) in the table we created 100 instances of both schemes. Then we attacked these instances by the UOV-attack to find out the number of trials we need to find a basis of $\mathcal{T}^{-1}(\mathcal{O})$. Table 5 shows the results.

		$(2^8, o, v, n)$	(8,16,24)	(10,20,30)	(12,24,36)	(16,32,48)	(20,40,60)
UOV	Rank(H)	n	9965	9963	9962	9966	9965
		n-1	35	37	38	34	35
		n-2	0	0	0	0	0
cyclicUOV	Rank(H)	n	9964	9957	9959	9964	9958
		n-1	0	0	0	0	0
		n-2	36	43	41	36	42

Table 4. Behavior of Rank attacks against our scheme

$(2^8$,o,v)	(5,7)	(8,11)	(12, 15)	(15, 18)
UOV	1734	531768	852738	1183621
cyclicUOV	1728	532614	847362	1146382

Table 5. Average number of trials in the UOV-attack

5 Example Parameters

Considering the above security analysis (especially the results of our experiments with rank attacks) and the fact, that a UOV scheme with $o = 24,\ v = 48$ is considered to be secure, we propose (for $K = GF(256)$) the parameters $o = 25,\ v = 50$.
Table 6 shows a comparison of the standard UOV scheme [KP99], the Rainbow signature scheme [DS05], [PB10] and our scheme.

	public key size (kB)	private key size(kB)	hashsize (bit)	signature size (bit)
UOV(44,48)	63.3	56.3	192	576
Rainbow(17,13,13)	25.7	19.1	208	344
cyclicUOV(25,50)	12.3	66.3	200	600

Table 6. Comparison of different UOV-based signature schemes

Besides the considerable reduction of the public key size, the number of multiplications needed in the verification process is decreased by about 41 %.
This can be seen as follows: To evaluate an arbitrary public key, for every quadratic term two K-multiplications are needed. Together with the n multiplications for the linear terms, one needs $n \cdot (n+2)$ multiplications for each polynomial. Hence, to evaluate the whole public key, one needs

$$o \cdot n \cdot (n+2)\ K - \text{multiplications} \tag{9}$$

When evaluating our partially cyclic public key, some of the multiplications can be used several times (For example, $b_1 \times x_1$ appears in every of the o public polynomials.) Thus, we do not have to carry out all the multiplications one by one. A close analysis shows, that by using this strategy we can reduce the number of multiplications needed in the verification process to

$$o \cdot n \cdot (n+2) - \left(\frac{n \cdot (n-1)}{2} - \frac{o \cdot (o-1)}{2}\right), \tag{10}$$

which, for $o = 25$ and $v = 50$, leads to a reduction of 41 %.

6 Conclusion and Future Work

We think that our proposal is an interesting idea how to reduce the public key size of multivariate schemes. Our preliminary security analysis seems to show that known attacks against UOV do not break our scheme. However, there remains a lot of work to be done. Some points we want to address in the future are

1. Completion of the security analysis of the present scheme (e.g. decomposition of polynomials)
2. Use of a pseudorandom matrix B
3. Extension of the idea to the Rainbow signature scheme [DS05]

7 Acknowledgements

We thank Jintai Ding, Christopher Wolf and Enrico Thomae for helpful comments and fruitful discussions.

References

[BB08] Bernstein, D.J., Buchmann, J., Dahmen, E. (Eds.): Post-Quantum Cryptography. Chapter Multivariate Cryptography. Springer, Heidelberg (2009)

[CC08] Chen, A.I.-T., Chen, C.-H. O., Chen, M.-S., Cheng, C.M., and Yang, B.-Y.: Practical-Sized Instances for Multivariate PKCs: Rainbow, TTS and ℓIC- Derivatives. In: LNCS 5299 pp. 95–108, Springer Heidelberg (2008)

[DG06] Ding, J., Gower, J. E., Schmidt, D.: Multivariate Public Key Cryptosystems. Springer, Heidelberg (2006)

[DS05] Ding J., Schmidt D.: Rainbow, a new multivariate polynomial signature scheme. In Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS vol. 3531, pp. 164–175 Springer, Heidelberg (2005)

[DY08] Ding, J., Yang, B.-Y., Chen, C.-H. O., Chen, M.-S., and Cheng, C.M.: New Differential-Algebraic Attacks and Reparametrization of Rainbow. In: LNCS 5037, pp.242–257, Springer, Heidelberg (2005)

[GC00] Goubin, L. and Courtois, N.T.: Cryptanalysis of the TTM cryptosystem. In Advances in Cryptology ASIACRYPT 2000, LNCS vol. 1976 , pp. 44–57. Tatsuaki Okamoto, ed., Springer (2000).

[KP99] Kipnis, A., Patarin, L., Goubin, L.: Unbalanced Oil and Vinegar Schemes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS vol. 1592, pp. 206–222 Springer, Heidelberg (1999)

[KS98] Kipnis, A., Shamir, A.: Cryptanalysis of the Oil and Vinegar Signature scheme. In: Krawzyck, H. (ed.) CRYPTO 1998, LNCS vol. 1462, pp. 257–266 Springer, Heidelberg (1998)

[Pa97] Patarin, J,: The oil and vinegar signature scheme, presented at the Dagstuhl Workshop on Cryptography (September 97)

[PB10] Petzoldt, A., Bulygin, S., Buchmann, J.: Selecting Parameters for the Rainbow Signature Scheme. To appear in: Proceedings of PQCrypto'10

[YC05] Yang, B.-Y., Chen J.-M.: Building secure tame like multivariate public-key cryptosystems: The new TTS. In: Boyd, C., Gonzales Nieto, J.M. (eds.) ACISP 2005. LNCS vol. 3574, pp. 518-531. Springer, Heidelberg (2005)

Multivariate Trapdoor Functions Based on Multivariate Left Quasigroups and Left Polynomial Quasigroups

S. Markovski[1], S. Samardziska[1], D. Gligoroski[2], S.J. Knapskog[3]

[1] Institute of Informatics, Faculty of Sciences, "Ss Cyril and Methodius" University, Skopje, Republic of Macedonia, {smile,simona}@ii.edu.mk
[2] Faculty for Informatics, Institute of Telematics, NTNU, Trondheim, Norway, danilog@item.ntnu.no
[3] Centre for Quantifiable Quality of Service in Communication Systems, NTNU, Trondheim, Norway, Svein.J.Knapskog@Q2S.ntnu.no

Abstract. A new class of multivariate quadratic trapdoor functions, MQQ, was defined elsewhere. The trapdoor functions were generated by quasigroup string transformations based on a class of quasigroups called multivariate quadratic quasigroups. The scheme was broken using Gröbner basis attacks and it can be used now only as a signature scheme. In order to prevent Gröbner basis attacks, in this paper we propose a modification of the previous scheme by mixing two types of multivariate functions. We have defined a family of s bit PKCs, for each $s \geq 104$. For $s = 160$ the public key produces 160-bit ciphertext by using 80 quadratic polynomials in 160 Boolean variables over $GF(2)$ and 10 polynomials in 20 variables over the ring $\mathbb{Z}_{256}$. All polynomials are obtained by left quasigroups. Using some mathematics, we conclude that our PKCs are infeasible on brute force and on nowadays known Gröbner bases attacks.

Key words: Public Key Cryptosystem, Left Quasigroup, Left Polynomial Quasigroup, Left Quasigroup String Transformation, Multivariate Quadratic Left Quasigroup

1 Introduction

The seminal paper of Diffie and Helman [2], where for the first time the public key paradigm was defined, has completely changed the way how we perceive the contemporary cryptography, and has influenced many areas of our modern information society. Nowadays ever growing e-commerce, e-banking, e-government, e-voting, etc. cannot be imagined without public key cryptsystems. There is a constant need for faster, more flexible and more secure public key designs. For example, the current e-voting systems would significantly gain on practicability and usability if the public key algorithms that they usually use (RSA [12] or Elliptic Curves Cryptography [7, 8]) would run faster (in the range of hundreds to thousand times faster).

Matsumoto and Imai [5] in 1985 defined the first multivariate quadratic (MQ) scheme, and up to 2005 three other types of MQ schemes were defined. All of these schemes were broken using different kinds of attacks. An excellent survey article for these four schemes is written by Wolf and Preneel [15].

Here we propose a new algorithm, called LQLP, that is aimed to overwhelm the weaknesses of the fifth MQ scheme, the MQQ algorithm, proposed by Gligoroski, Markovski and Knapskog in 2007 [3]. MQQ, which means multivariate quadratic quasigroup, was defined by using this new type of quasigroups. They allowed symbolic computations by quasigroups to be performed, and they were used for construction of multivariate quadratic polynomials as trapdoor functions. The MQQ algorithm, that is very fast in hardware as well as in software, was broken by Mohamed et al. in 2008 [9] and now can only be used for signature schemes.

The successful attacks on MQQ using Gröbner bases (or attacks of similar kinds) gave us the idea to produce a hybrid type of polynomials for trapdoor functions that are based on multivariate polynomials. For that purpose we use polynomials over the field $GF(2)$ and polynomials over the ring $\mathbb{Z}_{256}$ with the same set of Boolean variables. These polynomials are constructed using left multivariate quasigroups and left polynomial quasigroups. As much as we know, there have not been any applications of left quasigroups in cryptography (at least for symbolic computations) so far. Here we show how left quasigroups can be used for defining suitable trapdoor functions and we use them to define a new public key algorithm LQLP-s, where s is a positive integer denoting the bit length of the messages. Here we define the LQLP-160 in details, and we show how LQLP-128 and even LQLP-104 can be defined. We discuss that brute force and Gröbner bases attacks on LQLP-{104,128,160} are infeasible.

In section 2 we give a brief introduction to left quasigroups and to quasigroup string transformations, and we define new classes of Multivariate Quadratic Left Quasigroups (MQLQ) and Left Polynomial Quasigroups (LPQ). In section 3 we describe in details our PKC LQLP-160, based on MQLQs and LPQs. In Section 4 some mathematical properties of the system are considered. The operating characteristics are considered in Section 5, and the security aspects in Section 6. Conclusions are given in Section 7.

2 Left quasigroups

In this section we define left quasigroups, left quasigroup string transformations, presentation of the left quasigroups as vector valued Boolean functions and a construction of MQLQs and LPQs.

Definition 1. *A left quasigroup $(Q,*)$ is a groupoid, i.e., a nonempty set Q endowed with a binary operation $* : Q \times Q \to Q$, satisfying the law*

$$(\forall u, v \in Q)(\exists!\ x \in Q) \quad u * x = v. \tag{1}$$

It follows from (1) that for each $a, b \in Q$ there is a unique $x \in Q$ such that $a*x = b$. Also, the left cancellation law $a*x = a*y \implies x = y$ holds. The unique solution of the equation $a*x = b$ is denoted by $x = a \backslash_* b$ and then $\backslash_*$ is a binary operation on Q called a parastrophe (or adjoint operation) of $*$. The groupoid

$(Q, \backslash_*)$ is a left quasigroup too, since the element $x = a * b$ is the unique solution of the equation $a \backslash_* x = b$. In fact, the algebra $(Q, *, \backslash_*)$ satisfies the identities

$$x \backslash_* (x * y) = y, \quad x * (x \backslash_* y) = y. \tag{2}$$

Conversely, if the identities (2) hold true on $(Q, *, \backslash_*)$, then $(Q, *)$ is a left quasigroup and $\backslash_*$ is its parastrophe.

Left quasigroup string transformations. Consider an alphabet (i.e., a finite set) Q, and denote by Q^+ the set of all nonempty words (i.e., finite strings) formed by the elements of Q. In this paper, depending on the context, we will use two notifications for the elements of Q^+: $a_1a_2 \dots a_n$ and $(a_1, a_2, \dots, a_n)$, where $a_i \in Q$. Let $*$ be a left quasigroup operation on the set Q. For each $l \in Q$ we define two functions $e_{l,*}, d_{l,*} : Q^+ \to Q^+$ as follows.

Definition 2. *Let $a_i \in Q$, $M = a_1a_2 \dots a_n$. Then*
$e_{l,*}(M) = b_1b_2 \dots b_n \iff b_1 = l * a_1,\ b_2 = b_1 * a_2, \dots,\ b_n = b_{n-1} * a_n,$
$d_{l,*}(M) = c_1c_2 \dots c_n \iff c_1 = l * a_1,\ c_2 = a_1 * a_2, \dots,\ c_n = a_{n-1} * a_n,$
i.e., $b_{i+1} = b_i * a_{i+1}$ and $c_{i+1} = a_i * a_{i+1}$ for each $i = 0, 1, \dots, n-1$, where $b_0 = a_0 = l$.

The functions $e_{l,*}$ and $d_{l,*}$ are called respectively e–transformation and d–transformation of Q^+ based on the operation $*$ with leader l.

Theorem 1. *If $(Q, *)$ is a left quasigroup, then $e_{l,*}$ and $d_{l,\backslash_*}$ are mutually inverse permutations of Q^+, i.e., $d_{l,\backslash_*}(e_{l,*}(M)) = M = e_{l,*}(d_{l,\backslash_*}(M))$ for each leader $l \in Q$ and for every string $M \in Q^+$.*

Left quasigroups as vector valued Boolean functions. We will use the presentation of finite left quasigroups $(Q, *)$ of order 2^d by vector valued Boolean functions, where we take $Q = \{0, 1, \dots, 2^d - 1\}$ and we identify the elements of Q by their d-bit binary presentations. Now, the binary operation $*$ on Q can be viewed as a vector valued operation $*_{vv} : \{0,1\}^{2d} \to \{0,1\}^d$ defined as:

$$a * b = c \iff *_{vv}(x_1, x_2, \dots, x_d, y_1, y_2, \dots, y_d) = (z_1, z_2, \dots, z_d),$$

where $x_1 \dots x_d,\ y_1 \dots y_d,\ z_1 \dots z_d$ are binary presentations of a, b, c.

Each z_i depends of the bits $x_1, x_2, \dots, x_d, y_1, y_2, \dots, y_d$ and is uniquely determined by them. So, each z_i can be viewed as a $2d$-ary Boolean function $z_i = f_i(x_1, x_2, \dots, x_d, y_1, y_2, \dots, y_d)$, where $f_i : \{0,1\}^{2d} \to \{0,1\}$ strictly depends on and is uniquely determined by $*$. Thus, we have the following.

Lemma 1. *For every left quasigroup $(Q, *)$ of order 2^d there is a uniquely determined v.v.b.f. $*_{vv}$ and there are uniquely determined $2d$-ary Boolean functions $f_1, \dots, f_d$ such that for each $a, b, c \in Q$*

$$\begin{aligned} a * b = c \iff & *_{vv}(x_1, \dots, x_d, y_1, \dots, y_d) = \\ & = (f_1(x_1, \dots, x_d, y_1, \dots, y_d), \dots, f_d(x_1, \dots, x_d, y_1, \dots, y_d)). \end{aligned}$$

Recall that each k-ary Boolean function $f(x_1, \dots, x_k)$ can be presented in a unique way by its algebraic normal form (ANF), and we will not make difference between f and its ANF. We write $f(x_1, \dots, x_k)$ when we consider the arguments of a polynomial f to be the indeterminate variables $x_1, x_2, \dots, x_k$ of its ANF. The ANFs of the Boolean functions f_i give us information about the complexity of the left quasigroup $(Q, *)$ via the degrees of the polynomials f_i. Here we are interested in left quasigroups that have Boolean functions of degree 2. If, furthermore, they can be used for symbolic computations, such left quasigroups are said to be MQLQ (multivariate quadratic left quasigroups).

A construction of MQLQ. Here we present a construction of MQLQ that will be used in the sequel. Let $x_1, \dots, x_{2d}$ be Boolean variables, $d > 1$. Let $\mathbf{A_1} = [f_{ij}]_{d\times d}$ and $\mathbf{A_2} = [g_{ij}]_{d\times d}$ be two $d \times d$ nonsingular matrices of linear Boolean expressions, such that the functions f_{ij} and g_{ij} depend on the variables $x_1, \dots, x_d$. Let $\mathbf{D} = [d_{ij}]_{d\times d}$ be nonsingular Boolean matrix and let $\mathbf{c} = [c_i]_{d\times 1}$ be a Boolean vector. Then the following theorem holds.

Theorem 2. *The vector valued operation*

$$*_{vv}(x_1, \dots, x_{2d}) = \mathbf{D} \cdot (\mathbf{A_1} \cdot (x_{d+1}, \dots, x_{2d})^T + \mathbf{A_2} \cdot (x_1, \dots, x_d)^T + \mathbf{c}^T) \qquad (3)$$

defines a left quasigroup $(Q, *)$ *of order* 2^d *that is MQLQ, where* $Q = \{0, 1, \dots$ $\dots, 2^d - 1\}$*. The parastrophe* $\backslash_*$ *of* $*$ *is defined by*

$$\backslash_{*\,vv}(x_1, \dots, x_{2d}) = \mathbf{A_1}^{-1} \cdot (\mathbf{D}^{-1} \cdot (x_{d+1}, \dots, x_{2d})^T - \mathbf{A_2} \cdot (x_1, \dots, x_d)^T - \mathbf{c}^T).$$

Theorem 2 implies an easy construction of MQLQs. One only needs a suitable construction of the nonsingular matrices of linear Boolean expressions $\mathbf{A_1} = [f_{ij}]_{d\times d}$ and $\mathbf{A_2} = [g_{ij}]_{d\times d}$. A simple way to construct that kind of matrices is by the following *MQLQalgorithm*.

```
MQLQalgorithm
Input: Integer $d > 1$.
1. Generate uniformly at random $d(d-1)$ affine Boolean expressions denoted as
    $f_{1,2}, f_{1,3}, \dots, f_{1,d},\ f_{2,3}, f_{2,4}, \dots, f_{2,d},\ \dots,\ f_{d-2,d-1}, f_{d-2,d},\ f_{d-1,d},$
    $g_{1,2}, g_{1,3}, \dots, g_{1,d},\ g_{2,3}, g_{2,4}, \dots, g_{2,d},\ \dots,\ g_{d-2,d-1}, g_{d-2,d},\ g_{d-1,d};$
2. Define the terms $a_{i,j}$ of a $d \times d$-matrix $\mathbf{A_1}[a_{i,j}]$ as follows:
    $i > j \Longrightarrow a_{i,j} = 0, \quad a_{i,i} = 1, \quad i < j \Longrightarrow a_{i,j} = f_{i,j}$ ;
3. Define the terms $b_{i,j}$ of a $d \times d$-matrix $\mathbf{A_2}[b_{i,j}]$ as follows:
    $i > j \Longrightarrow b_{i,j} = 0, \quad b_{i,i} = 1, \quad i < j \Longrightarrow b_{i,j} = g_{i,j}$ ;
Output: A pair of $d \times d$ matrices $\mathbf{A_1}$ and $\mathbf{A_2}$ .
```

Left polynomial quasigroups. Let $P(x, y)$ be a bivariate polynomial over a ring $(R, +, \cdot)$. Define an operation $*$ on R by $a * b = P(a, b)$. If $(R, *)$ is a left quasigroup, then we say that P is a left polynomial quasigroup. In the sequel we are interested only of left polynomial quasigroups on the ring $(\mathbb{Z}_{2^n}, +, \cdot)$.

By a result of Rivest [13] we have the following simple criterion for a polynomial to be a left polynomial quasigroup.

Proposition 1. *A bivariate polynomial* $P(x, y)$ *over the ring* $(\mathbb{Z}_{2^n}, +, \cdot)$ *is a left polynomial quasigroup if and only if the univariate polynomials* $P(0, y)$ *and* $P(1, y)$ *are permutations.*

In the same paper Rivest gives a necessary and sufficient conditions a nonconstant polynomial $P(x) = a_0 + a_1x + a_2x^2 + \cdots + a_kx^k \in \mathbb{Z}_{2^n}[x]$ to be a permutation polynomial: a_1 is odd, $a_2 + a_4 + a_6 + \dots$ is even and $a_3 + a_5 + a_7 + \dots$ is even.

A bivariate polynomial $P(x, y)$ is of degree d if $d = m + n$ is the maximal number such that P contains a member ax^my^n, $a \neq 0$. We say that P is linear, quadratic, ternary, quarterly if $d = 1, 2, 3, 4$, respectively. Here we are interested up to quarterly polynomials.

Theorem 3. *Over the ring $(\mathbb{Z}_{2^n},+,\cdot)$ there are 2^{3n-1}, 2^{6n-3}, 2^{10n-5}, 2^{15n-3} bivariate linear, quadratic, ternary and quarterly polynomials, respectively, that define left polynomial quasigroups.*

Since different polynomials can define the same polynomial function, from the results given in [14], it can be shown that the following theorem holds.

Theorem 4. *Over the ring $(\mathbb{Z}_{2^n},+,\cdot)$, there are at least 2^{10n-15}, 2^{15n-25} bivariate ternary and quarterly left polynomial quasigroups.*

For $n=4$ there are at least $2^{25}, 2^{35}$, and for $n=8$ there are at least $2^{65}, 2^{95}$ bivariate ternary and quarterly left polynomial quasigroups, respectively.

The left quasigroup $(\mathbb{Z}_{2^n},*)$ defined by a left polynomial quasigroup $P(x,y)$ as $a*b=P(a,b)$, for each $a,b\in\mathbb{Z}_{2^n}$, has its parastrophe $\backslash_*$. It follows from the next theorem that that parastrophe is left polynomial quasigroup as well.

Theorem 5. *Let $P(x,y)$ be a left polynomial quasigroup over $\mathbb{Z}_{2^n}$. Then there is a left polynomial quasigroup $P_\backslash(x,y)$ such that*

$$P_\backslash(x,P(x,y)) = y = P(x,P_\backslash(x,y)).$$

The polynomial $P_\backslash$ over $\mathbb{Z}_{2^n}$ has, in general, very high degree, and it is impractical for effective computations. A much more effective way to compute the unknown x from the equation $P(a,x)=b$, where $a,b\in\mathbb{Z}_{2^n}$ are fixed, is by using the method of Hensel lifting [4]. It consists of finding the bits of the binary presentation of the unknown x by iteratively solving the equation $P(a,x)=b$ in the rings $\mathbb{Z}_2$ (the first bit can be found), $\mathbb{Z}_{2^2}$ (the second bit can be found), $\dots,\mathbb{Z}_{2^n}$ (the n-th bit can be found). Generally, it can happen the first bit to have two possible values (the bits 0 and 1), the second bit to have two possible values , and so on, i.e., in the process of finding the solution a branching can appear. For the case of permutation polynomials we have the following result.

Theorem 6. *Let $P(x)=a_0+a_1x+a_2x^2+\cdots+a_dx^d$ be a permutation polynomial over $\mathbb{Z}_{2^n}$, and let $b\in\mathbb{Z}_{2^n}$. Then, the equation $P(x)=b$ over $\mathbb{Z}_{2^n}$ can be solved using Hensel lifting in exactly n steps without any branching, i.e., the i-th bit of the unique solution x is unambiguously determined in the i-th step.*

3 Description of the algorithm LQLP-160

For our cryptsystem LQLP-160 we use the *MLQLPolyQ* algorithm, defined by two auxiliary algorithms *MQLQequations* and *LPolyQequations*. We also need another two auxiliary algorithms *InverseMQLQ* and *InverseLPolyQ* for the decryption phase. These algorithms are given below.

MQLQequations
Input: 80 affine Boolean functions $f_1,f_2,\dots,f_{80}$ such that each f_i depends on 160 Boolean variables $x_1,x_2,\dots,x_{160}$.
1. Represent a vector $x=(f_1,\dots,f_{80})$ of affine Bool. functions of variables $x_1,\dots,x_{160}$ as a string $x=X_1\dots X_{16}$ of vectors X_i of dimension 5; 2. Generate at random 16 left quasigroups of order 2^5, $(Q,*_1),\dots,(Q,*_{16})$ by using the *MQLQalgorithm*;

```
3. Transform by using $d$-transformations the string $x = X_1 \dots X_{16}$ into the string
   $y = Y_1 \dots Y_{16}$, where $Y_i$ are vectors of dimension 5, as follows:
   Take a random vector $L \in \{0,1\}^5$ as leader and put
     $Y_1 = L *_1 X_1,\ Y_2 = X_1 *_2 X_2,\ Y_3 = X_2 *_3 X_3, \dots,\ Y_{16} = X_{15} *_{16} X_{16}$;
4. Generate uniformly at random a nonsingular $80 \times 80$ Boolean matrix $S_{80}$ and
   represent the string $y = Y_1 Y_2 \dots Y_{16}$ as a vector $y = (y_1, y_2, \dots, y_{80})$;
5. Compute $S_{80} \cdot y^T = (p_1, p_2, \dots, p_{80})$;
Output: 80 MQ polynomials $p_i(x_1, x_2, \dots, x_{160})$ over the field $GF(2)$.
```

```
$InverseMQLQ$
Input: A vector $b$ of 80 bits $b = (b_1, b_2, \dots, b_{80})$.
1. Compute $S_{80}^{-1} \cdot b^T = (y_1, y_2, \dots, y_{80}) = y$;
2. Represent the vector $y$ as a string $y = Y_1 \dots Y_{16}$ of vectors $Y_i$ of dimension 5;
3. Transform by using $e$-transformations the string $y = Y_1 \dots Y_{16}$ into the string
   $f' = F'_1 \dots F'_{16}$, where $F'_i$ are vectors of bits of dimension 5, as follows:
     $F'_1 = L \backslash_{*_1} Y_1,\ F'_2 = F'_1 \backslash_{*_2} Y_2,\ F'_3 = F'_2 \backslash_{*_3} Y_3, \dots,\ F'_{16} = F'_{15} \backslash_{*_{16}} Y_{16},$ ;
Output: 80 bits $f'_1, \dots, f'_{80}$, where $F'_i = (f'_{5i-4}, \dots, f'_{5i})$ for $i = 1, \dots, 16$.
```

```
$LPolyQequations$
Input: 20 byte variables $C_1, C_2, \dots, C_{10}, D_1, D_2, \dots, D_{10}$ such that
       each $C_i$ and each $D_i$ depends on Boolean variables $x_1, x_2, \dots, x_{160}$.
1. Choose uniformly at random 3 nonsingular $10 \times 10$ matrices $S'_{10}$, $S''_{10}$
   and $S'''_{10}$ over the ring $\mathbb{Z}_{256}$;
2. Choose randomly a constant vector $const = (const_1, \dots, const_{20}) \in \mathbb{Z}_{256}^{20}$;
3. Compute $\begin{bmatrix} S'_{10} & O \\ S''_{10} & S'''_{10} \end{bmatrix} \cdot (C_1, \dots, C_{10}, D_1, \dots, D_{10})^T + const^T =$
   $= (C'_1, \dots, C'_{10}, D'_1, \dots, D'_{10})$;
4. Choose uniformly at random a sequence of 10 left polynomial quasigroups
   of degree 3 $P_1(x,y), P_2(x,y), \dots, P_{10}(x,y) \in \mathbb{Z}_{256}[x,y]$ ;
5. Denote by $(q'_1, q'_2, \dots, q'_{10})$ the vector $(P_1(C'_1, D'_1), \dots, P_{10}(C'_{10}, D'_{10}))$;
6. Choose uniformly at random a nonsingular $10 \times 10$ matrix $S_{10}$ over the
   ring $\mathbb{Z}_{256}$ and compute $S_{10} \cdot (q'_1, q'_2, \dots, q'_{10})^T = (q_1, q_2, \dots, q_{10})$;
Output: 10 multivariate polynomials $q_i(C_1, C_2, \dots, C_{10}, D_1, D_2, \dots, D_{10})$
        over the ring $\mathbb{Z}_{256}$.
```

```
$InverseLPolyQ$
Input: 20 bytes $B_1, \dots, B_{10},\ C_1, \dots, C_{10}$.
1. Compute $S_{10}^{-1} \cdot (B_1, B_2, \dots, B_{10})^T = (B'_1, \dots, B'_{10})$;
2. Compute $S'_{10} \cdot (C_1, \dots, C_{10})^T + (const_1, \dots, const_{10})^T = (C'_1, \dots, C'_{10})$;
3. Denote by $P_{i\backslash}(x,y)$ the parastrophe of $P_i(x,y)$,   $i = 1, 2, \dots, 10$;
4. Denote by $(D'_1, \dots, D'_{10})$ the vector $(P_{1\backslash}(C'_1, B'_1), \dots, P_{10\backslash}(C'_{10}, B'_{10}))$;
5. Compute $(D_1, \dots, D_k) =$
     $S_{10}'''^{-1} \cdot ((D'_1, \dots, D'_{10})^T - (const_{11}, \dots, const_{20})^T - S''_{10} \cdot (C_1, \dots, C_{10})^T)$;
Output: 10 bytes $D_1, \dots, D_{10}$.
```

```
$MLQLPolyQ$
Input: 160 Boolean variables $x_1, x_2, \dots x_{160}$.
1. Generate uniformly at random a nonsingular $160 \times 160$ Boolean matrix $S$;
2. Denote by $(f_1, f_2, \dots, f_{160})$ the vector $S \cdot (x_1, x_2, \dots, x_{160})^T$;
3. Compute $MQLQequations(f_1, f_2, \dots, f_{80}) =$
   $(p_1(x_1, \dots, x_{160}), p_2(x_1, \dots, x_{160}), \dots, p_{80}(x_1, \dots, x_{160}))$;
```

```
4. Choose randomly two nonsingular $80 \times 80$ Boolean matrices $S_L$ and $S_R$;
5. Compute $S_L \cdot (f_1, f_2, \dots, f_{80})^T = (y_1, y_2, \dots, y_{80})$ and
   $S_R \cdot (f_{81}, f_{82}, \dots, f_{160})^T = (y_{81}, y_{82}, \dots, y_{160})$ ;
6. Construct a vector of 10 bytes $(C_1, \dots, C_{10})$ as
   $C_1 = y_1||y_2||\dots||y_8,\ C_2 = y_9||y_{10}||\dots||y_{16},\ \dots,\ C_{10} = y_{73}||y_{74}||\dots||y_{80}$;
7. Construct a vector of 10 bytes $(D_1, \dots, D_{10})$ as
   $D_1 = y_{81}||y_{82}||\dots||y_{88},\ D_2 = y_{89}||\dots||y_{96}, \dots, D_{10} = y_{153}||y_{154}||\dots||y_{160}$;
8. Compute $LPolyQequations(C_1, \dots, C_{10}, D_1, \dots, D_{10}) =$
       $= (q_1(C_1, \dots, C_{10}, D_1, \dots, D_{10}), \dots, q_{10}(C_1, \dots, C_{10}, D_1, \dots, D_{10}))$;
Output: 80 MQ polynomials $p_i(x_1, x_2, \dots, x_{160})$ over $GF(2)$
and 10 multivariate polynomials $q_i(C_1, \dots, C_{10}, D_1, \dots, D_{10})$ over $\mathbb{Z}_{256}$.
```

Creation of a private and a public key. Let $x = (x_1, x_2, \dots, x_{160})$ be a vector of 160 Boolean variables. The creation of the public and the private key is obtained by application of the $MLQLPolyQ$ algorithm.

The public key consists of 80 equations over $GF(2)$, 10 equations over $\mathbb{Z}_{256}$, and a 160×160 matrix $A = S'S$, where $S' = \begin{bmatrix} S_L & O \\ O & S_R \end{bmatrix}$. The 90 equations are as follows:

$$\begin{gathered} b_1 = p_1(x_1, x_2, \dots, x_{160}), \dots, b_{80} = p_{80}(x_1, x_2, \dots, x_{160}), \\ B_1 = q_1(C_1, \dots, C_{10}, D_1, \dots, D_{10}), \dots, B_{10} = q_{10}(C_1, \dots, C_{10}, D_1, \dots, D_{10}). \end{gathered} \tag{4}$$

Note that, for given b_i and B_j, (4) is a system of 90 equations with 160 Boolean unknowns $x_1, \dots, x_{160}$. Namely, as defined, C_i and D_i are functions of $y_1, y_2, \dots, y_{160}$, and each y_i is a function of $x_1, x_2, \dots, x_{160}$ as well.

The private key consists of the following tuple: $(S, S_{80}, S_L, S_R, S_{10}, S'_{10}, S''_{10}, S'''_{10}, *_1, \dots, *_{16}, L, const, P_1(x, y), \dots, P_{10}(x, y)$, where $S,\ S_{80},\ S_L,\ S_R$ are nonsingular Boolean matrices of dimensions $160 \times 160,\ 80 \times 80,\ 80 \times 80,\ 80 \times 80$, respectively, while $S_{10},\ S'_{10},\ S''_{10},\ S'''_{10}$ are nonsingular matrices of dimension 10×10 over $\mathbb{Z}_{256}$, $*_1, \dots, *_{16}$ are left quasigroup operations of order 2^5, $L \in \{0, 1\}^5$, $const \in \mathbb{Z}_{256}^{20}$ and $P_i(x, y) \in \mathbb{Z}_{256}[x, y]$ are polynomial quasigroups.

The flowchart of the creation of the public key is presented in Figures 1.

Encryption. The encryption is performed by using the matrix A and the system of equations (4). To encrypt a message $M = m_1 m_2 \dots m_{160}$ consisting of 160 bits m_i, we first evaluate the polynomials $p_1, p_2, \dots, p_{80}$ and we get 80 bits $b_1, b_2, \dots, b_{80}$, where $b_i = p_i(m_1, m_2, \dots, m_{160})$. Next we compute $A \cdot (m_1, m_2, \dots, m_{160})^T = (m'_1, m'_2, \dots, m'_{160})$ and we form 20 bytes $C_1, \dots, C_{10}$, $D_1, \dots, D_{10}$ as $C_1 = m'_1||m'_2||\dots||m'_8,\ \dots,\ C_{10} = m'_{73}||m'_{74}||\dots||m'_{80},\ D_1 = m'_{81}||m'_{82}||\dots|||m'_{88},\ \dots,\ D_{10} = m'_{153}||m'_{154}||\dots||m'_{160}$. Then we evaluate the polynomials $q_1, q_2, \dots, q_{10}$ and we get 10 bytes $B_1, B_2, \dots, B_{10}$, where $B_i = q_i(C_1, \dots, C_{10}, D_1, \dots, D_{10})$.

The ciphertext is an 160 bits string $b_1 \dots b_{80}||B_1||\dots||B_{10}$.

The flowchart of the encryption is presented in Figure 2.

Decryption and Signature Generation. Let us have a ciphertext $b_1 b_2 \dots b_{160}$, where b_i are bits. The decryption procedure is as follows.

1. Compute $(f'_1, f'_2, \dots, f'_{80}) = InverseMQLQ(b_1, b_2, \dots, b_{80})$;

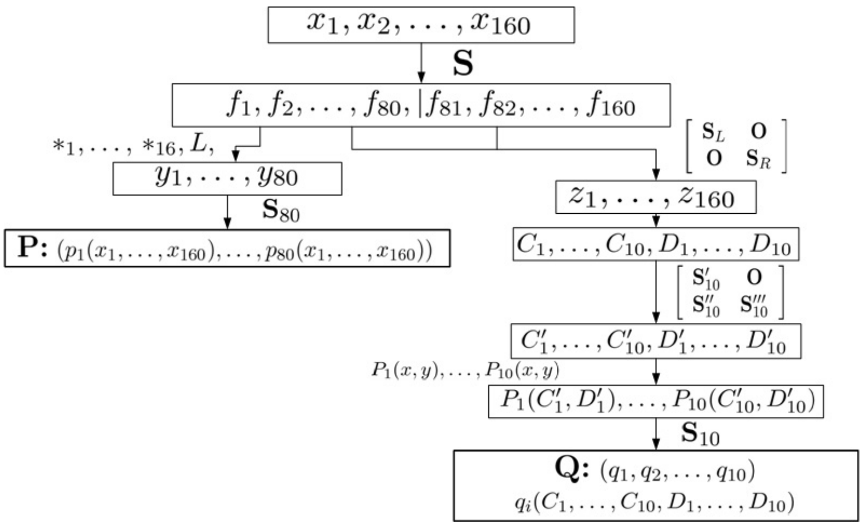

$x_1, x_2, \ldots, x_{160}$
$\mathbf{S}$
$f_1, f_2, \ldots, f_{80}, | f_{81}, f_{82}, \ldots, f_{160}$
$*_1, \ldots, *_{16}, L,$
$\begin{bmatrix} \mathbf{S}_L & \mathbf{O} \\ \mathbf{O} & \mathbf{S}_R \end{bmatrix}$
$y_1, \ldots, y_{80}$
$\mathbf{S}_{80}$
$z_1, \ldots, z_{160}$
P: $(p_1(x_1, \ldots, x_{160}), \ldots, p_{80}(x_1, \ldots, x_{160}))$
$C_1, \ldots, C_{10}, D_1, \ldots, D_{10}$
$\begin{bmatrix} \mathbf{S}'_{10} & \mathbf{O} \\ \mathbf{S}''_{10} & \mathbf{S}'''_{10} \end{bmatrix}$
$C'_1, \ldots, C'_{10}, D'_1, \ldots, D'_{10}$
$P_1(x,y), \ldots, P_{10}(x,y)$
$P_1(C'_1, D'_1), \ldots, P_{10}(C'_{10}, D'_{10})$
$\mathbf{S}_{10}$
Q: $(q_1, q_2, \ldots, q_{10})$
$q_i(C_1, \ldots, C_{10}, D_1, \ldots, D_{10})$

Fig. 1. Creation of the public key.

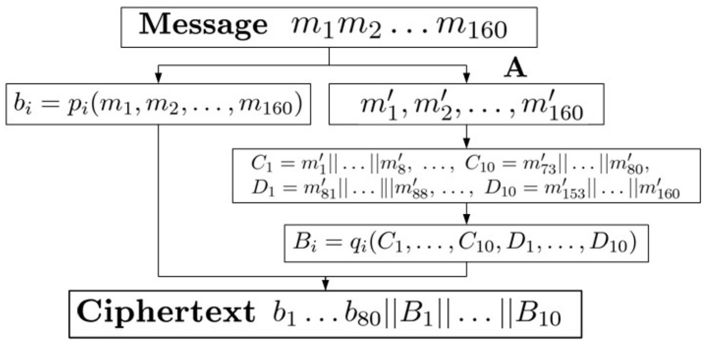

Message $m_1 m_2 \ldots m_{160}$
A
$b_i = p_i(m_1, m_2, \ldots, m_{160})$
$m'_1, m'_2, \ldots, m'_{160}$
$C_1 = m'_1 || \ldots || m'_8, \ \ldots, \ C_{10} = m'_{73} || \ldots || m'_{80},$
$D_1 = m'_{81} || \ldots ||| m'_{88}, \ \ldots, \ D_{10} = m'_{153} || \ldots || m'_{160}$
$B_i = q_i(C_1, \ldots, C_{10}, D_1, \ldots, D_{10})$
Ciphertext $b_1 \ldots b_{80} || B_1 || \ldots || B_{10}$

Fig. 2. Encryption.

2. Construct a vector of 10 bytes $(B_1, \ldots, B_{10})$, as $B_1 = b_{81}||b_{82}||\ldots||b_{88}$, $B_2 = b_{89}||b_{90}||\ldots||b_{96}, \ \ldots, \ B_{10} = b_{153}||b_{154}||\ldots||b_{160}$;
3. Compute $(y_1, \ldots, y_{80}) = S_L \cdot (f'_1, f'_2, \ldots, f'_{80})^T$;
4. Construct a vector of 10 bytes $(C_1, \ldots, C_{10})$ as $C_1 = y_1||\ldots||y_8, \ \ldots, \ C_{10} = y_{73}||\ldots||y_{80}$;
5. Compute $(D_1, \ldots, D_{10}) = InverseLPolyQ(C_1, \ldots, C_{10}, B_1, \ldots, B_{10})$;
6. Represent the string of bytes $D_1||D_2||\ldots||D_{10}$ as a string of bits $y_{81} \ldots y_{160}$;
7. Compute $(f'_{81}, f'_{82}, \ldots, f'_{160}) = S_R^{-1} \cdot (y_{81}, y_{82}, \ldots, y_{160})^T$;
8. Compute $(x_1, x_2, \ldots, x_{160}) = S^{-1} \cdot (f'_1, f'_2, \ldots, f'_{160})^T$;
9. The obtained string $x_1 x_2 \ldots x_{160}$ of 160 bits is the plaintext.

The flowchart of the decryption is presented in Figure 3.

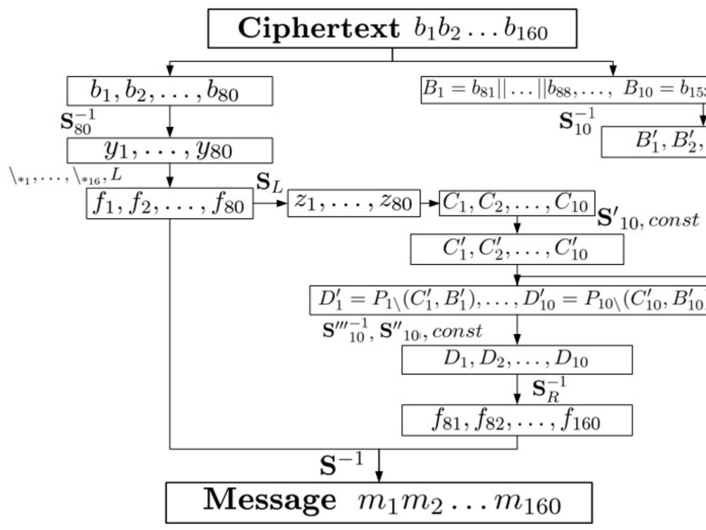

Ciphertext $b_1b_2\dots b_{160}$
$b_1, b_2, \dots, b_{80}$
$B_1 = b_{81}||\dots||b_{88}, \dots, B_{10} = b_{153}||\dots||b_{160}$
$\mathbf{S}_{80}^{-1}$
$\mathbf{S}_{10}^{-1}$
$y_1, \dots, y_{80}$
$B'_1, B'_2, \dots, B'_{10}$
$\backslash_{*_1}, \dots, \backslash_{*_{16}}, L$
$\mathbf{S}_L$
$f_1, f_2, \dots, f_{80}$
$z_1, \dots, z_{80}$
$C_1, C_2, \dots, C_{10}$
$\mathbf{S}'_{10}, const$
$C'_1, C'_2, \dots, C'_{10}$
$D'_1 = P_1\backslash(C'_1, B'_1), \dots, D'_{10} = P_{10}\backslash(C'_{10}, B'_{10})$
$\mathbf{S}'''^{-1}_{10}, \mathbf{S}''_{10}, const$
$D_1, D_2, \dots, D_{10}$
$\mathbf{S}_R^{-1}$
$f_{81}, f_{82}, \dots, f_{160}$
$\mathbf{S}^{-1}$
Message $m_1m_2\dots m_{160}$

Fig. 3. Decryption.

Using the private key, the signature of a message M is performed in a standard way, i.e., we first apply a 160 bit hash function H on the message M and then we apply the decryption procedure on the string $H(M)$.

4 Some mathematical properties of the algorithm

We consider some mathematical properties of LQLP-160. Let denote by $E,\ D : \{0,1\}^{160} \to \{0,1\}^{160}$ the encryption (E) and the decryption (D) function. These functions make use of the properties of the left quasigroups, in the sense that they have adjoint operations that satisfy the identities (2). From that, and from Theorem 1, the next theorem follows.

Theorem 7. *The functions E and D are permutations on the set $\{0,1\}^{160}$.*

One very important consequence of Theorem 7 is the next one, that follows from the fact that the function D is a bijection.

Theorem 8. *Let $b_1, \dots, b_{80}$ be given bits and let $B_1, \dots, B_{10}$ be given bytes. Then the system of equations* (4) *has unique solution $(x_1, \dots, x_{160}) \in \{0,1\}^{160}$.*

Next, we discuss the system of equations (4). It consists of 80 equations

$$b_1 = p_1(x_1, x_2, \dots, x_{160}), \dots, b_{80} = p_{80}(x_1, x_2, \dots, x_{160}), \tag{5}$$

over the field $GF(2)$, and 10 equations

$$B_1 = q_1(C_1, \dots, C_{10}, D_1, \dots, D_{10}), \dots, B_{10} = q_{10}(C_1, \dots, C_{10}, D_1, \dots, D_{10}), \tag{6}$$

over the ring $(\mathbb{Z}_{256}, +, \cdot)$. Here, b_i are given bits and B_j are given bytes.

From the equations (6) we can extract 80 equations with Boolean variables y_i as it is shown below.

In the system (6) we replace each C_i by $2^7 y_{8i-7} + \cdots + 2^2 y_{8i-2} + 2y_{8i-1} + y_{8i}$ and each D_i by $2^7 y_{80+8i-7} + \cdots + 2y_{80+8i-1} + y_{80+8i}$, for $i = 1, \ldots, 10$. After rearrangement, the polynomials q_j for $j = 1, 2, \ldots, 10$, can be expressed as

$$q_j = 2^7 q_{j,7} + 2^6 q_{j,6} + 2^5 q_{j,5} + 2^4 q_{j,4} + 2^3 q_{j,3} + 2^2 q_{j,2} + 2q_{j,1} + q_{j,0}, \qquad (7)$$

where $q_{j,i}$ are multivariate polynomials on $\mathbb{Z}_{256}(+, \cdot)$ with variables $y_1, \ldots, y_{160}$ and with coefficients 0 or 1.

Proposition 2. *The polynomials $q_{j,0}, q_{j,1}, \ldots, q_{j,7}$ in* (7) *are uniquely determined.*

Let denote by $q^{\oplus}_{j,i}$ the polynomial over $(\mathbb{Z}_2, \oplus, \cdot)$ obtained from the polynomial $q_{j,i}$ over $(\mathbb{Z}_{256}, +, \cdot)$ by replacing all occurrences of the operation $+$ by $\oplus$.

Next, let denote by $t_{j,7}t_{j,6}t_{j,5}t_{j,4}t_{j,3}t_{j,2}t_{j,1}t_{j,0}$ the binary representation of q_j, where $t_{j,i}$ are Boolean variables. We are going to represent the bits $t_{j,i}$ buy using the bits $q^{\oplus}_{j,i}$ and some additional Boolean variables.

For the binary representation $t_{j,0,7}t_{j,0,6}t_{j,0,5}t_{j,0,4}t_{j,0,3}t_{j,0,2}t_{j,0,1}t_{j,0,0}$ of $q_{j,0}$, where $t_{j,0,r}$ are Boolean variables, we have

$$t_{j,0} = t_{j,0,0} = q^{\oplus}_{j,0} \qquad (8)$$

and then q_j can be represented as

$$q_j = 2^7(q_{j,7} + t_{j,0,7}) + \ldots 2^2(q_{j,2} + t_{j,0,2}) + 2(q_{j,1} + t_{j,0,1}) + q^{\oplus}_{j,0}. \qquad (9)$$

Now, let denote by $t_{j,1,7}t_{j,1,6}t_{j,1,5}t_{j,1,4}t_{j,1,3}t_{j,1,2}t_{j,1,1}t_{j,1,0}$ the binary representation of $q_{j,1} + t_{j,0,1}$, where $t_{j,1,r}$ are Boolean variables. Then we have that

$$t_{j,1} = t_{j,1,0} = q^{\oplus}_{j,1} \oplus t_{j,0,1} \qquad (10)$$

and q_j can be represented as

$$q_j = 2^7(q_{j,7}+t_{j,0,7}+t_{j,1,7})+\cdots+2^2(q_{j,2}+t_{j,0,2}+t_{j,1,2})+2(q^{\oplus}_{j,1}\oplus t_{j,0,1})+q^{\oplus}_{j,0}. \qquad (11)$$

In the same way we have

$$t_{j,2} = t_{j,2,0} = q^{\oplus}_{j,2} \oplus t_{j,0,2} \oplus t_{j,1,2} \qquad (12)$$

and q_j can be represented as

$$q_j = 2^7(q_{j,7}+t_{j,0,7}+t_{j,1,7}+t_{j,2,7})+\cdots+2^2(q^{\oplus}_{j,2}\oplus t_{j,0,1}\oplus t_{j,1,2})+2(q^{\oplus}_{j,1}\oplus t_{j,0,1})+q^{\oplus}_{j,0}, \qquad (13)$$

where $t_{j,2,7}t_{j,2,6}\ldots t_{j,1,1}t_{j,1,0}$ is the binary representation of $q_{j,2} + t_{j,0,2} + t_{j,1,2}$.

Finally, continuing in the same manner of indexing the variables of type t, after 8 steps we have that

$$q_j = 2^7(q_{j,7}\oplus t_{j,0,7}\oplus\ldots t_{j,6,7})+\cdots+2^2(q^{\oplus}_{j,2}\oplus t_{j,0,1}\oplus t_{j,1,2})+2(q^{\oplus}_{j,1}\oplus t_{j,0,1})+q^{\oplus}_{j,0}, \qquad (14)$$

and

$$\begin{aligned}
t_{j,3} &= t_{j,3,0} = q^{\oplus}_{j,3} \oplus t_{j,0,3} \oplus t_{j,1,3} \oplus t_{j,2,3},\\
t_{j,4} &= t_{j,4,0} = q^{\oplus}_{j,4} \oplus t_{j,0,4} \oplus t_{j,1,4} \oplus t_{j,2,4} \oplus t_{j,3,4},\\
t_{j,5} &= t_{j,5,0} = q^{\oplus}_{j,5} \oplus t_{j,0,5} \oplus t_{j,1,5} \oplus t_{j,2,5} \oplus t_{j,3,5} \oplus t_{j,4,5},\\
t_{j,6} &= t_{j,6,0} = q^{\oplus}_{j,6} \oplus t_{j,0,6} \oplus t_{j,1,6} \oplus t_{j,2,6} \oplus t_{j,3,6} \oplus t_{j,4,6} \oplus t_{j,5,6},\\
t_{j,7} &= t_{j,7,0} = q^{\oplus}_{j,7} \oplus t_{j,0,7} \oplus t_{j,1,7} \oplus t_{j,2,7} \oplus t_{j,3,7} \oplus t_{j,4,7} \oplus t_{j,5,7} \oplus t_{j,6,7}.
\end{aligned} \qquad (15)$$

We have the following lemma.

Lemma 2. *Let denote the binary representation of the bytes B_j by $b_{j,7}b_{j,6}b_{j,5}$ $b_{j,4}b_{j,3}b_{j,2}b_{j,1}b_{j,0}$ for $j = 1, 2, \ldots, 10$. Then any solution $(y'_1, y'_2, \ldots, y'_{160}) \in \{0,1\}^{160}$ of the system of 10 equations (6) over the ring $(\mathbb{Z}_{256}, +, \cdot)$ is also a solution of the following system of 80 equations over the field $(\mathbb{Z}_2, \oplus, \cdot)$, where $\overline{y} = (y_1, y_2, \ldots, y_{160})$*

$$\begin{array}{llll} b_{1,0} = q^{\oplus}_{1,0}(\overline{y}), & b_{1,1} = q^{\oplus}_{1,1}(\overline{y}) \oplus u_{1,1}, & \ldots, & b_{1,7} = q^{\oplus}_{1,7}(\overline{y}) \oplus u_{1,7}, \\ b_{2,0} = q^{\oplus}_{2,0}(\overline{y}), & b_{2,1} = q^{\oplus}_{2,1}(\overline{y}) \oplus u_{2,1}, & \ldots, & b_{2,7} = q^{\oplus}_{2,7}(\overline{y}) \oplus u_{2,7}, \\ \multicolumn{4}{l}{\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots} \\ b_{10,0} = q^{\oplus}_{10,0}(\overline{y}), & b_{10,1} = q^{\oplus}_{10,1}(\overline{y}) \oplus u_{10,1}, & \ldots, & b_{10,7} = q^{\oplus}_{10,7}(\overline{y}) \oplus u_{10,7}, \end{array} \tag{16}$$

for suitable choice of the Boolean variables $u_{i,j}$ $(i = 1, 2, \ldots, 10,\ j = 0, 1, 2, \ldots, 7)$.

We have taken the symbols y_i in (16) to represent Boolean variables. In our algorithms we have in fact that y_i are linear Boolean expressions with variables $x_1, x_2, \ldots, x_{160}$, i.e., $y_i = y_i(x_1, x_2, \ldots, x_{160})$. After replacing the variables y_i by its linear expressions $y_i(x_1, x_2, \ldots, x_{160})$, from (16) we obtain the following system of 80 equations on the field $\mathbb{Z}_2(\oplus, \cdot)$, where $\overline{x} = (x_1, x_2, \ldots, x_{160})$

$$\begin{array}{llll} b_{1,0} = s_{1,0}(\overline{x}), & b_{1,1} = s_{1,1}(\overline{x}) \oplus u_{1,1}, & \ldots, & b_{1,7} = s_{1,7}(\overline{x}) \oplus u_{1,7}, \\ b_{2,0} = s_{2,0}(\overline{x}), & b_{2,1} = s_{2,1}(\overline{x}) \oplus u_{2,1}, & \ldots, & b_{2,7} = s_{2,7}(\overline{x}) \oplus u_{2,7}, \\ \multicolumn{4}{l}{\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots} \\ b_{10,0} = s_{10,0}(\overline{x}), & b_{10,1} = s_{10,1}(\overline{x}) \oplus u_{10,1}, & \ldots, & b_{10,7} = s_{10,7}(\overline{x}) \oplus u_{10,7}, \end{array} \tag{17}$$

where $s_{j,i}(x_1, x_2, \ldots, x_{160}) = q^{\oplus}_{j,i}(y_1(\overline{x}), y_2(\overline{x}), \ldots, y_{160}(\overline{x}))$.

From Lemma 2 we have the following theorem.

Theorem 9. *Let the system of 10 equations (6) on the ring $(\mathbb{Z}_{256}, +, \cdot)$ depend on the variables $x_1, x_2, \ldots, x_{160}$. Then any solution $x'_1, x'_2, \ldots, x'_{160} \in \{0,1\}^{160}$ of (6) gives a solution $(x'_1, x'_2, \ldots, x'_{160}, u'_{1,1}, u'_{1,2}, \ldots, u'_{10,7}) \in \{0,1\}^{230}$ of (17), for suitably chosen bits $u'_{j,i}$ for the unknowns $u_{j,i}$.*

Theorem 10. *There is only one solution $(x'_1, x'_2, \ldots, x'_{160}, u'_{1,1}, u'_{1,2}, \ldots, u'_{10,7}) \in \{0,1\}^{230}$ of the system of 160 equations with 230 Boolean variables, obtained by joining the systems (5) and (17), such that $(x'_1, x'_2, \ldots, x'_{160}) \in \{0,1\}^{160}$ is a solution of the system of equations (4).*

5 Operating characteristics of LQLP-160

In this section we will discuss the size of the private and the public key as well as the number of operations per byte for encryption and decryption.

The size of the public and the private key. For a message block of 160 bits the memory needed for the 80 multivariate quadratic equations over $GF(2)$ is $80(1 + \binom{160}{1} + \binom{160}{2}) = 1030480$ bits, or less than 126 Kbytes. For the 10 equations over $\mathbb{Z}_{256}$ we need $10(1 + \binom{20}{1} + \binom{20}{2} + \binom{20}{3}) = 13510$ bytes, or less than 14 Kbytes. For the matrix A we need less than 4 Kbytes. Altogether, the public key can be stored in 143 Kbytes.

The private key consists of the Boolean matrices S, S_{80}, S_L, S_R, of the matrices of bytes $S_{10}, S'_{10}, S''_{10}, S'''_{10}$, of a bit vector L and of a byte vector *const*, that can be stored in less than 5.9 Kbytes. Each of the 16 MQLQs requires 0.01 Kbytes. Finally, for the polynomials $P_i(x, y) \in \mathbb{Z}_{256}[x, y]$ of degree 3, we need 0.1 Kbytes. Altogether, the public key can be stored in 6 Kbytes.

Number of operations for encryption and decryption. We assume that AND, XOR, addition and multiplication over $\mathbb{Z}_{256}$ are executed in one cycle.

A rough non-optimized estimate of the number of operations for the encryption can be summarized in two formulas for the evaluation of the MQ polynomials over $GF(2)$ and polynomial of degree 3 over $\mathbb{Z}_{256}$, respectively. Thus, 3078400 bit operations are needed for 80 MQ polynomials and 50600 byte operations are needed for 10 polynomials over $\mathbb{Z}_{256}$.

In the decryption procedure we have 3 matrix-vector multiplications of dimension 80 in bits (36660 bit operations), one matrix-vector multiplication of dimension 160 (51400 bit operations), 4 matrix-vector multiplications with 3 subtractions of dimension 10 in bytes (790 byte operations). For 16 left quasigroups we have 14732 bit operations, and for 10 Hensel liftings in the polynomials $P_i(x, y)$ we have 12780 bit operations.

A very rough non-optimized estimate of the cycles per encrypted/decrypted byte on 8, 32 and 64-bit architectures is given in Table 1.

Cycles per encrypted byte			Cycles per decrypted byte		
Arch=8-bit	Arch=32-bit	Arch=64-bit	Arch=8-bit	Arch=32-bit	Arch=64-bit
21770	5443	2722	761	191	96

Table 1. Non-optimized estimate of cycles per encrypted/decrypted byte

Parallelization. We do not consider here the problem of parallelization of our algorithms. Still, we note that many of the computations can be made in parallel. All of the evaluations of the polynomials p_i, $i = 1, 2, \ldots, 80$, and of the polynomials P_i, $i = 1, 2, \ldots, 10$, as well as the left quasigroups $*_i$, $i = 1, 2, \ldots, 16$, can be done in parallel, since they can be realized independently. Also, that can be done for the matrix computations, an for some other actions, too. In parallel implementations, the estimates of Table 1 will be significantly smaller.

6 Security analysis of the algorithm

We will address several security aspects of LQLP-160. From the first two subsections we conclude that the brute force attack is computationally infeasible. In the fourth subsection we show that a Gröbner bases attack has a complexity higher than 2^{80}. Some other kinds of attacks are briefly mentioned in the subsequent subsections.

The size of the pool of MQLQ quasigroups and of LPQ. The MQLQs of order 2^5, as constructed in Section 2 by the *MQLQalgorithm*, are build from two 5×5-matrices $\mathbf{A}_1$ and $\mathbf{A}_2$, each one containing 10 affine Boolean expressions with variables x_1, x_2, x_3, x_4, x_5. Since there are $2^6 = 64$ such affine Boolean expressions, there are 2^{120} different constructions of pairs of such matrices. Also, we need an 5×5 nonsingular Boolean matrix $\mathbf{D}$ and a vector $\mathbf{c} \in \{0,1\}^5$. Since around 28% of randomly generated Boolean matrices are nonsingular, there are approximately $0.28 \cdot 2^{5^2} = 2^{23}$ matrices $\mathbf{D}$. So, there are $2^{120} 2^{23} 2^5 2^4 = 2^{152}$ different constructions of 16 left quasigroups of order 2^5. Clearly, not all of them may define different left quasigroups. Anyhow, the pool of left quasigroups of order 2^5, defined by *MQQalgorithm*, is a huge one. (Note that the number of left quasigroups of order 2^5 is $32!^{32}$.)

By Theorem 4 there are at least 2^{10n-15} ternary left polynomial quasigroups over the ring $\mathbf{Z}_{2^n}$. So, 10 left polynomial ternary quasigroups over the ring $\mathbf{Z}_{2^8}$ can be chosen in 2^{650} different ways.

Decomposition of the public matrix A. The public key contains a Boolean 160×160 matrix A, obtained as a product of the block matrix $S' = \begin{bmatrix} S_L & O \\ O & S_R \end{bmatrix}$ and the matrix S, i.e., $A = S'S$. S_L and S_R are Boolean 80×80 matrices, while S is a nonsingular Boolean 160×160 matrix. An attack can be performed if the matrices S and S' are known. We show that, given the matrix A, it is computationally infeasible to find the matrices S and S'. Namely, the following property holds.

Proposition 3. *Let A be a given nonsingular Boolean 160×160 matrix, and let X, Y and Z be unknown nonsingular Boolean matrices, such that X and Y are of dimension 80×80, while Z is of dimension 160×160. Then the matrix equation $\begin{bmatrix} X & O \\ O & Y \end{bmatrix} \cdot Z = A$ has as many solutions as there are pairs of nonsingular Boolean 80×80 matrices, i.e., approximately $(0.28 \cdot 2^{80^2})^2 > 2^{12796}$ solutions.*

Gröbner bases attack. In order to prevent our PKC from Gröbner bases attack we use different types of polynomial equations. For starting an attack, one has to transform the polynomials over the ring $\mathbf{Z}_{256}$ into multivariate polynomials over the field $GF(2)$. A way of how a transformation can be done is shown in Section 4, where a system of 160 multivariate polynomials with 230 variables is obtained. So, the system has 2^{70} different solutions, and by Theorem 10 exactly one of them gives the plaintext. Since the complexity of finding a solution of a system of 160 multivariate (at least quadratic) equations with 160 unknowns in $GF(2)$ is much larger than 2^{10}, we conclude that a Gröbner bases attack on our PKC of this type has a complexity at least 2^{80}.

An attacker can form a much simpler system of 90 multivariate polynomial equations in $GF(2)$ with 160 unknowns as follows. Take the 80 polynomials from (5) and only the following 10 polynomials

$$b_{1,0} = q^{\oplus}_{1,0}(\overline{y}), \quad b_{2,0} = q^{\oplus}_{2,0}(\overline{y}), \quad \ldots, \quad b_{10,0} = q^{\oplus}_{10,0}(\overline{y}),$$

from (16), where a new variable $u_{i,j}$ is not involved. Anyhow, 2^{70} solutions, i.e., application of Gröbner bases is needed again.

An attacker may try to develop a system in $GF(2)$ directly from the system (6) by using the fact that $x+y = x \oplus y + 2xy$, where x, y are bits, $+$ is operation in $\mathbb{Z}_{256}$ and $\oplus$ is operation in $\mathbb{Z}_2$. In that case the attacker will obtain 80 equations in 160 variables on $GF(2)$ of degree at least 13. One equation will consist in average of 2^{60} terms. As much as we know, the system of 160 equations in 160 variables of degree at least 13 of this kind, cannot be solved by Gröbner bases.

Since the XL [1] procedure for solving MQ polynomials [6] is equivalent to Gröbner bases attack, our PKC is resistent for XL attacks too.

Isomorphism of polynomials and MinRank attacks. The isomorphism of polynomials with one secret introduced by Patarin [10] can be briefly formulated as follows. For two multivariate mappings $P, P' : \{0,1\}^n \to \{0,1\}^n$ given by their polynomials $(P_1(x_1,\dots,x_n),\dots,P_n(x_1,\dots,x_n))$ and $(P'_1(x_1,\dots,x_n),\dots,$ $P'_n(x_1,\dots,x_n))$ find (if any) an invertible affine mapping $S : \{0,1\}^n \to \{0,1\}^n$ such that $P' = S \circ P$, where the operation $\circ$ is composition of mappings. Using isomorphism of polynomials, Peret in 2005 [11] showed that the scheme of Patarin is not secure. We cannot see how this attack can be realized on our PKC, especially since the inner polynomials are not known, so the attack cannot be mount.

The MinRank problem is as follows. Given a sequence of matrices $(M_1,\dots,$ $M_n)$ over some field and an integer $r < n$, find a linear combination of the matrices such that $Rank(\sum_{i=1}^{n} \lambda_i M_i) \leq r$. The MinRank attacks consists of representation of a given public key of n polynomials $P_i(x_1,\dots,x_n)$ in the form $P_i(x_1,\dots,x_n) = x^T M_i x$, where M_i are $n \times n$ matrices in a field. If the private key is constructed by polynomials $P'_i(x_1,\dots,x_n)$ such that they can be described as $P'_i(x_1,\dots,x_n) = x^T A_i x$, and if the minimal rank r of A_i is much smaller than n, then there are effective algorithms for finding linear combination of the matrices M_i such that $Rank(\sum_{i=1}^{n} \lambda_i M_i) \leq r$. The information for those linear combinations can be used for breaking the system. We cannot see how these attacks can be realized on our PKC, having in mind that the obtained system of equations contains polynomials of degree at least 3. On the other hand, the MQLQs can be chosen such that the minimal ranks of their quadratic polynomial, when represented in matrix form, is at least 8.

7 Conclusion

By using left quasigroups that allow symbolic computations we have designed a PKC LQLP-160, in details. The design of LQLP-160 can be easily modified for LQLP-s, $s \geq 104$, in such a way to be still resistent against brute force and Gröbner bases attacks. What we need is to keep the 10 polynomial equations (6) while reducing the number of polynomial equations (5). Some technical slight modifications of the algorithms *MLQLPolyQ*, *MQLQequations* and *LPolyQequations* should be made, as well. Thus, for LQLP-128 we have to separate the 128 binary variables x_i into two groups: $x_1,\dots,x_{48}$ (for building MQLQs) and $x_{49},\dots,x_{128}$ (for building LPQs). Then, for example, we can modify the line 6. of $MLQLPolyQ$ algorithm by using 48 variables as follows: "Construct a vector of 10 bytes $(C_1, \dots, C_{10})$ as $C_1 = y_1 || \dots || y_8, \dots,\ C_6 = y_{41} || \dots || y_{48},\ C_7 = y_{13} || \dots || y_{20},\ C_8 = y_{21} || \dots || y_{28},\ C_9 = y_{29} || \dots || y_{36},\ C_{10} = y_{37} || \dots || y_{44}$." Also, the matrix S_L is of dimension 48×48, while S_R is still of dimension 80×80, and the matrix S_{80} has to be replaced by a matrix S_{48}. Instead of 16 MQLQ of order 2^5 we can use 8 MQLQ of order 2^6, and so on.

We have taken $s \geq 104$ since 104 bits are still enough a secure LQLP algorithm to be constructed, by using 24 bit variables only for building MQLQs.

We have succeeded to construct a PKC with a moderate speed, comparable to today's standard PKCs. Its advantages are the lower number of bits variables, i.e., the length of the messages, and the possibility to be realized in parallel. The parallelization can make our PKC much faster in hardware implementations.

References

1. N. Courtois, A. Klimov, J. Patarin, and A. Shamir: *Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations*, in B. Preenel, editor, Advances in Cryptology–Eurocrypt 2000, LNCS 1807/2000, Springer, pp. 392–407.
2. W. Diffie and M. Hellman: *New Directions in Cryptography*, IEEE Trans. Inform. Theory, Vol IT-22, No 6, (1976), 644–654.
3. D. Gligoroski, S. Markovski, and S.J. Knapskog: *Multivariate quadratic trapdoor functions based on multivariate quadratic quasigroups*, American Conference on Applied Mathematics; Harvard, March 2008, USA.
4. E. Kaltofen: *Sparse Hensel Lifting*, EUROCAL'85, European Conf. Comput. Algebra Proc. Vol. 2, 1985, pp. 4–17.
5. Imai, H. and Matsumoto T.: *Algebraic methods for constructing asymmetric cryptosysytems*, in Proceedings of 3rd Intern. Conf. AAECC-3, Grenoble, France, July 15-19, 1985, J. Calmet ed., LNCS 29/1985, Springer, pp. 108–119.
6. A. Kipnis, A. Shamir: *Cryptanalysis of the HFE public key cryptosystem*, In Advances in Cryptology, CRYPTO 1999, LNCS 1666/1999, Springer, pp. 19–30.
7. N. Koblitz: *Elliptic curve cryptosystems*, in Mathematics of Computation 48, (1987), pp. 203–209.
8. V. Miller: *Use of elliptic curves in cryptography*, CRYPTO 85, 1985.
9. M. S. E. Mohamed, J. Ding, J. Buchmann and F. Werner *Algebraic Attack on the MQQ Public Key Cryptosystem*, LNCS 5888/2009, Springer, pp. 392–401.
10. J. Patarin: *Hidden Field Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of asymmetric algorithms*, in Advances in Cryptology, EUROCRYPT 1996, LNCS 1070/1996, Springer, pp. 33–48.
11. L. Peret: A Fast Cryptanalysis of the Isomorphism of Polynomials with One Secret Problem, EUROCRYPT 2005, LNCS 3494, Springer, pp. 354-370, 2005.
12. R. Rivest, A. Shamir and L. Adleman: *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*, Comm. ACM, Vol 21, No 2, (1978), pp. 120–126.
13. Ronald L. Rivest: *Permutation polynomials modulo* 2^w, Finite Fields and Their Applications 7, (2001) pp. 287–292
14. S. Samardziska: *Polynomial quasigroups of order* p^w, Masters' thesis, Skopje, 2009. `http://sites.google.com/site/samardziska`
15. C. Wolf and B. Preneel: *Taxonomy of Public Key Schemes based on the problem of Multivariate Quadratic equations*, Cryptology ePrint Archive, Report 2005/077, 2005.