

Refined Application of the XL Algorithm within the Algebraic Approach to the Regular Syndrome Decoding Problem

Hiroki Furue^{1*} and Yusuke Aikawa²

^{1*}NTT Social Informatics Laboratories, 3-9-11, Midoricho,
Musashino-shi, 180-8585, Tokyo, Japan.

²The University of Tokyo, 7-3-1, Hongo, Bunkyo-ku, 113-8654, Tokyo,
Japan.

*Corresponding author(s). E-mail(s): hiroki.furue@ntt.com;
Contributing authors: aikawa@mist.i.u-tokyo.ac.jp;

Abstract

The Syndrome Decoding Problem (SDP) is a fundamental mathematical problem underlying code-based cryptography, which is considered a promising candidate for post-quantum cryptography. The Regular Syndrome Decoding Problem (RSDP) is a variant of the SDP in which the error vector follows a specific block-wise structure. Recent developments in RSDP-based cryptographic schemes have led to a wider variety of parameter settings, making it increasingly important to evaluate the complexity of solving the RSDP over a broad range of parameters. One of the main approaches to solving the RSDP is the algebraic approach, which transforms a given RSDP instance into a polynomial system and solves it using algebraic techniques such as the XL algorithm. In this work, we construct and analyze algebraic algorithms for the RSDP by combining and extending techniques used in existing algebraic approaches. Our algorithms mainly follow the framework of the hybrid algorithm proposed by Wang et al., while allowing an arbitrary solving degree d in the application of the XL algorithm. We further consider two variants: an XFL-based method that combines partial Gaussian elimination with variable guessing, and a WXL-based method that exploits the sparsity of the Macaulay matrix. Our complexity analysis for various parameter settings shows that, for some fixed values of n and k , the WXL-based variant using a larger solving degree $d > 2$ is particularly efficient when the block size β is sufficiently small, while the original hybrid algorithm becomes more efficient as β increases.

Keywords: post-quantum cryptography, code-based cryptography, syndrome decoding problem, regular syndrome decoding problem, XL algorithm

1 Introduction

The Syndrome Decoding Problem (SDP) is a fundamental computational problem in coding theory and can be formulated as follows. Given a parity-check matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$ of a linear code of length n and dimension k , a syndrome $\mathbf{s} \in \mathbb{F}_q^{n-k}$, and a target weight h , the SDP asks to find an error vector $\mathbf{e} \in \mathbb{F}_q^n$ such that $\mathbf{H}\mathbf{e} = \mathbf{s}$ and the weight of $\mathbf{e}$ equals h . Code-based cryptography is a family of public-key cryptosystems whose security is based on the hardness of SDP, and it is regarded as a promising approach to post-quantum cryptography (PQC). Indeed, the code-based key-encapsulation mechanism HQC [1] was recently selected for standardization in the NIST PQC project [2]. Therefore, analyzing the computational hardness of the SDP is an important research problem from both theoretical and practical perspectives.

This paper addresses a variant of the SDP, referred to as the Regular Syndrome Decoding Problem (RSDP). The RSDP was introduced by Augot et al. in 2005 [3] as the security assumption underlying the Fast Syndrome-Based hash functions. The difference between RSDP and SDP is that the RSDP restricts the error vector $\mathbf{e}$ to a regular form, where $\mathbf{e}$ consists of multiple equally sized blocks, each having exactly one nonzero element. A number of RSDP-based cryptographic schemes, including Multi-Party Computation (MPC) [4] and Pseudorandom Correlation Generator (PCG) [5], have been proposed. These schemes achieve improved efficiency by leveraging the “regular” structure of the RSDP. Furthermore, RSDP-based signature schemes constructed within the frameworks of MPC or VOLE-in-the-head have recently been proposed [6–8] and have attracted significant attention.

These developments in RSDP-based cryptographic schemes indicate that it is important to evaluate the complexity of solving the RSDP over a broad range of parameters, including different code lengths, dimensions, block sizes, and error weights. Previous concrete complexity analyses have mainly focused on parameter sets motivated by PCG constructions [5]. Meanwhile, some recently proposed RSDP-based schemes, particularly signature schemes based on the MPC-in-the-head and VOLE-in-the-head paradigms [6–8], consider a wider variety of parameter settings. Accordingly, investigating a wider variety of parameter regimes is essential for understanding the behavior of existing attacks and for accurately assessing the security of both existing and future RSDP-based constructions.

Initially, the complexity of solving the RSDP was estimated by directly applying algorithms developed for the plain SDP [4, 5]. However, several algorithms have recently been proposed specifically for the RSDP, which solve the problem more efficiently by exploiting its regular structure. The algorithms for the RSDP can be roughly classified into two groups: The first group consists of approaches based on information set decoding (ISD) [9, 10], which integrate ideas from combinatorics and linear algebra. The second group consists of algebraic approaches [11–13] that transform a

given instance into a polynomial system. In this paper, we focus on these algebraic approaches and discuss the possibility of further improving their efficiency.

Before reviewing the individual algebraic algorithms for the RSDP, we briefly introduce the basic method for solving multivariate polynomial systems employed in these algorithms, namely the XL algorithm [14]. Given a solving degree d , the XL algorithm first generates equations of degree at most d by multiplying the input equations by monomials of appropriate degrees. It then constructs the corresponding coefficient matrix, called the Macaulay matrix, and applies Gaussian elimination to this matrix. If the solving degree d is sufficiently large, the elimination produces sufficiently simplified equations from which a solution to the polynomial system can be recovered.

At EUROCRYPT 2023, Briaud and Øygarden [11] proposed the first algebraic algorithm for the RSDP. This attack treats each element of the error vector $\mathbf{e} = (e_1, \dots, e_n)$ as a variable. For these n variables, they consider two types of equations: (1) $n - k$ linear equations derived from $\mathbf{H}\mathbf{e} = \mathbf{s}$ and (2) quadratic equations of the form $e_i \cdot e_j = 0$, where e_i and e_j belong to the same block, imposed by the regular structure of $\mathbf{e}$. They solve the system consisting of these two types of equations by applying the XL algorithm.

Wang et al. [13] also proposed an algebraic method for solving the RSDP, known as the hybrid algorithm. This algorithm also considers the two types of equations (1) and (2) as in [11]. Unlike the original algebraic algorithm, it eliminates $n - k$ variables from the quadratic equations (2) using the linear equations (1) before solving the system. The hybrid algorithm then algebraically solves the resulting quadratic system with fewer variables by applying the BDT algorithm [15]. The BDT algorithm can be regarded as the special case $d = 2$ of the Crossbred algorithm [16], which combines the guessing of a subset of variables with preprocessing of the Macaulay matrix. Furthermore, Wang et al. refined the strategy for guessing variables, in order to further reduce complexity. As a result, they demonstrated that their proposed algorithm outperforms previous algorithms for the RSDP, particularly in the case of large finite fields and large error weights.

More recently, Liu et al. [12] proposed another algebraic algorithm for the RSDP, called AGB2.0. Their algorithm mainly modifies the variable-guessing strategy of the algebraic algorithm proposed by Briaud and Øygarden [11]. Unlike the hybrid algorithm, AGB2.0 does not use the linear equations (1) to obtain a quadratic system with fewer variables. Instead, it solves the resulting system of linear equations (1) and quadratic equations (2) by constructing a Macaulay matrix with an arbitrary solving degree d .

The hybrid algorithm proposed by Wang et al. is known to achieve the lowest complexity among existing algorithms for a wide range of parameters. One possible reason for its efficiency is that, unlike other algebraic algorithms, it uses the linear equations to eliminate variables from the quadratic equations, thereby reducing the number of variables in the resulting quadratic system. However, the hybrid algorithm considers only the case $d = 2$, meaning that the degree of the resulting quadratic equations is not increased. Therefore, investigating whether the efficiency of the hybrid algorithm can be further improved by removing this restriction on the solving degree is an important research problem.

1.1 Our Contribution

In this work, we construct and analyze algebraic algorithms for the RSDP by refining the way the XL algorithm is applied to the polynomial system obtained from existing algebraic approaches. The proposed algorithms mainly follow the framework of the hybrid algorithm introduced by Wang et al. [13]. Specifically, we first construct the quadratic system by removing some variables from the quadratic equations (2) using the linear equations (1). We then solve the resulting quadratic system with fewer variables by applying the XL algorithm.

The main difference from the original hybrid algorithm lies in the choice of the solving degree d used in the XL algorithm. Unlike the original hybrid algorithm, we allow an arbitrary value for the solving degree. This generalization of d broadens the range of feasible parameters for the algorithm. Moreover, for higher solving degrees, the Macaulay matrix exhibits greater sparsity, which can be exploited to further improve efficiency.

Furthermore, we consider two variants for applying the XL algorithm to the resulting quadratic equations. Throughout the algorithm, we guess a subset of variables following the refined strategy used in the hybrid algorithm. In [13], the BDT algorithm partially performs Gaussian elimination on the Macaulay matrix before guessing some variables in the quadratic system to reduce the complexity. We extend this technique to the case of an arbitrary solving degree d , leading to the first variant of our algorithm. More specifically, we propose a method based on the XFL algorithm [14, 17], which partially eliminates the Macaulay matrix before the guessing step. In contrast, the second variant is based on the Wiedemann XL (WXL) algorithm [18]. This variant does not perform any preprocessing before the guessing step, but exploits the sparsity of the Macaulay matrix using the Wiedemann algorithm [19].

In this paper, we present a method for estimating the complexity of both variants of the proposed algorithm. In this estimation, we use the Hilbert series introduced in [11] to estimate the solving degree. Furthermore, we provide a structural reinterpretation of this Hilbert series by focusing on the linear dependencies among the equations generated during the attack.

For the proposed algorithms, we first evaluate the complexity for the reference parameters used in [9–11, 13, 20] (see Table 2 in Subsection 5.1 below). As a result, we confirmed that the XFL-based algorithm with $d = 2$, which corresponds to the original hybrid algorithm, is the best algorithm for most of the reference parameters. Furthermore, we evaluate the complexity in settings where the block size β is small, meaning that the weight of $\mathbf{e}$ is large (see Tables 3 and 4 in Subsection 5.2 below). These complexity analyses reveal that, for some fixed values of n and k , the proposed WXL-based variant achieves the lowest complexity when β is sufficiently small. As β increases, however, the original hybrid algorithm eventually achieves the lowest complexity. For some parameter settings with small block sizes, the proposed XFL-based variant with a solving degree $d > 3$ also achieves competitive complexity. For example, when $n = 2^{16}$ and $k = 2048$, the WXL-based variant achieves the lowest complexity for $\beta = 5, 10, 15$, whereas the XFL-based variant with $d = 3$ achieves the lowest complexity for $\beta = 20$. These results demonstrate that the proposed generalization

of the solving degree can reduce the complexity of algebraic attacks in small-block-size settings that have not been sufficiently explored in previous studies. This finding may have implications for parameter selection in future RSDP-based cryptographic schemes.

Organization.

The rest of this paper is organized as follows: Section 2 introduces the notation and recalls the RSDP, its algebraic modeling, and the XL algorithm. Section 3 presents the proposed extensions of the hybrid algorithm. Section 4 estimates the solving degree and the complexity of the proposed algorithms. Section 5 evaluates the complexity of the algorithms for specific parameters. Section 6 discusses the asymptotic complexity of the proposed algorithms. Section 7 concludes the paper and summarizes the key findings.

2 Preliminaries

Subsection 2.1 introduces the notation. Subsection 2.2 reviews the RSDP and Subsection 2.3 explains the algebraic modeling of the problem. Subsection 2.4 provides an overview of the XL algorithm for solving multivariate polynomial systems.

2.1 Notation

Let $\mathbb{F}_q$ denote the finite field with q elements. We generally use column vectors and omit the transposition symbol $\top$ for simplicity. For two column vectors $\mathbf{a} \in \mathbb{F}_q^m$ and $\mathbf{b} \in \mathbb{F}_q^n$, we write their concatenation as $(\mathbf{a}, \mathbf{b}) \in \mathbb{F}_q^{m+n}$. For a vector $\mathbf{a} \in \mathbb{F}_q^m$, a_i denotes the i -th coordinate of $\mathbf{a}$, and $\text{wt}(\mathbf{a}) = |\{i \mid a_i \neq 0\}|$ denotes the number of nonzero elements in $\mathbf{a}$. The space of n -by- m matrices over $\mathbb{F}_q$ is denoted by $\mathbb{F}_q^{n \times m}$, and the n -by- n identity matrix is denoted by $\mathbf{I}_n$. For a positive integer n , we denote by $[n]$ the set $\{1, \dots, n\}$.

2.2 Regular Syndrome Decoding Problem

This subsection reviews the definitions of the original and regular syndrome decoding problems.

A linear code $\mathcal{C}$ of length n and dimension k with $k < n$ over the finite field $\mathbb{F}_q$ is defined as a k -dimensional vector subspace of $\mathbb{F}_q^n$. For such a code, a parity-check matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$ is defined such that its kernel equals $\mathcal{C}$. In the decoding process, the goal is to recover the codeword $\mathbf{c} \in \mathcal{C}$ from a received message $\mathbf{c}' = \mathbf{c} + \mathbf{e}$ affected by an error vector $\mathbf{e}$. Syndrome decoding is a method to recover the error term $\mathbf{e}$ from $\mathbf{H}\mathbf{c}' = \mathbf{H}(\mathbf{c} + \mathbf{e}) = \mathbf{H}\mathbf{e}$.

Definition 1 (Syndrome Decoding Problem) For four positive integers (q, n, k, h) with $k < n$ and $h < n$, given an $(n - k)$ -by- n matrix $\mathbf{H}$ over $\mathbb{F}_q$ and $\mathbf{s} \in \mathbb{F}_q^{n-k}$, the syndrome decoding problem asks to find a vector $\mathbf{e} \in \mathbb{F}_q^n$ such that $\mathbf{H}\mathbf{e} = \mathbf{s}$ and $\text{wt}(\mathbf{e})$ is equal to h .

We then consider a variant of the SDP, which is called the regular syndrome decoding problem (RSDP). This RSDP was introduced by Augot et al. [3] in 2005 as the underlying assumption for the Fast Syndrome-Based hash function. The difference between SDP and RSDP is only the form of the error vector $\mathbf{e}$. In the RSDP, $\mathbf{e}$ is partitioned into equally sized blocks, and each block contains exactly one nonzero element, which is called the regular structure. We then write down the RSDP as follows:

Definition 2 (Regular Syndrome Decoding Problem) Let (q, n, k, h) be four positive integers such that $k < n$ and $h \mid n$, and let $\beta = n/h$. Given an $(n - k)$ -by- n matrix $\mathbf{H}$ over $\mathbb{F}_q$ and $\mathbf{s} \in \mathbb{F}_q^{n-k}$, the regular syndrome decoding problem asks to find a vector $\mathbf{e} = (\mathbf{e}_1, \dots, \mathbf{e}_h) \in \mathbb{F}_q^n$ with $\mathbf{e}_1, \dots, \mathbf{e}_h \in \mathbb{F}_q^\beta$ such that $\mathbf{H}\mathbf{e} = \mathbf{s}$ and $\text{wt}(\mathbf{e}_i) = 1$ for $i \in [h]$.

Note that, for convenience, the parameters of the RSDP are sometimes represented in the form (q, n, k, β) in certain cases in this paper.

2.3 Algebraic Modeling

This subsection reviews the algebraic modeling for the RSDP provided in [11]. In this modeling, each element of the error vector $\mathbf{e} = (e_1, \dots, e_n)$ is regarded as a variable. We also represent these n variables in the following form

$$(e_{1,1}, \dots, e_{1,\beta}, \dots, e_{h,1}, \dots, e_{h,\beta}),$$

where each $e_{k,i}$ corresponds to the i -th element of the k -th block $\mathbf{e}_k$. From the definition of the problem, we can directly derive $n - k$ linear equations in the form of

$$\mathbf{H}\mathbf{e} = \mathbf{s}.$$

Furthermore, from the regular structure of $\mathbf{e}$, we can construct the following quadratic equations

$$e_{k,i} \cdot e_{k,j} = 0 \quad (k \in [h], 1 \leq i < j \leq \beta). \quad (1)$$

Consequently, a solution to the RSDP can be found by solving the following polynomial system

$$\begin{cases} \mathbf{H}\mathbf{e} = \mathbf{s} \\ e_{k,i} \cdot e_{k,j} = 0 \quad (k \in [h], 1 \leq i < j \leq \beta) \end{cases}. \quad (2)$$

This system consists of $n - k$ linear equations and $h \cdot \binom{\beta}{2}$ quadratic equations. We consider applying the XL algorithm [14] explained in Subsection 2.4 below to solve the resulting system. In [11], additional equations are provided for the case of $q = 2$. However, in this paper, we restrict our attention to the case of larger finite fields and therefore focus on the system defined by (2).

2.4 XL Algorithm

This subsection briefly reviews the XL algorithm for solving multivariate polynomial systems over finite fields. The XL algorithm, proposed by Courtois et al. [14], is the

most fundamental linearization-based method. Its basic idea is simple: linearizing the given system by treating each monomial as one variable. Suppose that we are given multivariate polynomials $f_1, \dots, f_m$ in n variables $\mathbf{x} = (x_1, \dots, x_n)$ over $\mathbb{F}_q$ with $\deg f_i = d_i$. We then try to find a solution $\mathbf{x}$ such that $f_1(\mathbf{x}) = \dots = f_m(\mathbf{x}) = 0$. For a chosen degree d , the algorithm multiplies every polynomial f_i by monomials of degree $\leq d - d_i$ in $\mathbf{x}$ to generate equations of degree $\leq d$. We then construct a coefficient matrix, which is called the Macaulay matrix, whose rows and columns correspond to the obtained polynomials of degree $\leq d$ and the monomials of degree $\leq d$, respectively. If the degree d is sufficiently large and a sufficient number of equations are prepared, a univariate equation can be obtained by performing Gaussian elimination on the Macaulay matrix. We then solve the obtained univariate equation, and the same procedure is repeated for the remaining variables.

Example 1 (XL algorithm) Consider the following three quadratic polynomials in two variables x_1 and x_2 over $\mathbb{F}_7$:

$$\begin{aligned} f_1 &= 6x_1x_2 + 3x_2^2 + x_1 + 2, \\ f_2 &= 3x_1^2 + 6x_1x_2 + 2x_1 + 5x_2 + 4, \\ f_3 &= 5x_1^2 + 3x_1x_2 + 4x_2^2 + 3x_1 + x_2 + 1. \end{aligned}$$

We aim to solve the equations $f_i(x_1, x_2) = 0$ with $1 \leq i \leq 3$ by applying the XL algorithm with degree $d = 3$. The corresponding Macaulay matrix is constructed as follows:

$$\begin{array}{c} x_1^3 \quad x_1^2x_2 \quad x_1x_2^2 \quad x_2^3 \quad x_1^2 \quad x_1x_2 \quad x_2^2 \quad x_1 \quad x_2 \quad 1 \\ \begin{matrix} x_1f_1 \\ x_1f_2 \\ x_1f_3 \\ x_2f_1 \\ x_2f_2 \\ x_2f_3 \\ f_1 \\ f_2 \\ f_3 \end{matrix} \end{array} \begin{pmatrix} 0 & 6 & 3 & 0 & 1 & 0 & 0 & 2 & 0 & 0 \\ 3 & 6 & 0 & 0 & 2 & 5 & 0 & 4 & 0 & 0 \\ 5 & 3 & 4 & 0 & 3 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 6 & 3 & 0 & 1 & 0 & 0 & 2 & 0 \\ 0 & 3 & 6 & 0 & 0 & 2 & 5 & 0 & 4 & 0 \\ 0 & 5 & 3 & 4 & 0 & 3 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6 & 3 & 1 & 0 & 2 \\ 0 & 0 & 0 & 0 & 3 & 6 & 0 & 2 & 5 & 4 \\ 0 & 0 & 0 & 0 & 5 & 3 & 4 & 3 & 1 & 1 \end{pmatrix}.$$

Gaussian elimination is then applied to transform the matrix into the following form

$$\begin{array}{c} x_1^3 \quad x_1^2x_2 \quad x_1x_2^2 \quad x_2^3 \quad x_1^2 \quad x_1x_2 \quad x_2^2 \quad x_1 \quad x_2 \quad 1 \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 6 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 4 \end{pmatrix} \end{array}.$$

As a result, univariate equations can then be obtained, from which the solution $(x_1, x_2) = (4, 3)$ is found.

As mentioned above, the XL algorithm finds a solution when the chosen degree d is sufficiently large. We refer to the minimum degree at which the algorithm succeeds as the solving degree. Several results exist for estimating the degree using the Hilbert series under certain assumptions, depending on the form of the input polynomials [11, 17, 21–23]. In particular, we discuss how to estimate the solving degree for the polynomial systems obtained through the algebraic approach to the RSDP in Subsection 4.1 below.

One efficient approach that applies the XL algorithm is to combine it with an exhaustive search [24, 25]. In this approach, a subset of variables is fixed, and the remaining system is solved using the XL algorithm. This process is iterated until a solution is found. In the proposed algorithms presented in Section 3, we adopt this idea to solve the multivariate system derived from the RSDP, following the framework of existing algebraic algorithms.

3 Proposed Algorithms for RSDP

This section describes the algorithms for the RSDP considered in this work, constructed based on the hybrid algorithm proposed in [13]. These algorithms consist of two main steps, outlined as follows:

1. Transform a given instance of the RSDP into a system of quadratic equations (in Subsection 3.1).
2. Solve the resulting system by applying the XL algorithm (in Subsections 3.2 and 3.3).

The main difference from the hybrid algorithm [13] is that we allow an arbitrary solving degree $d \geq 2$ in the second step. Moreover, we solve the resulting polynomial systems over the quotient ring modulo the relations of the form (1), as in [11]. Within this framework, we present two methods for applying the XL algorithm to the resulting polynomial system: The first method, explained in Subsection 3.2, is based on the XFL algorithm [14, 17], and the second, explained in Subsection 3.3, is based on the Wiedemann XL (WXL) algorithm [18]. As in the hybrid algorithm, we use three nonnegative integers (u, f, g) as parameters satisfying the following conditions

$$\begin{aligned} u &\leq \beta - 2, \\ f + g &\leq h, \\ (u + 1)f + u(h - g - f) + \beta g &< k. \end{aligned}$$

These parameters are set to adjust the number of guessed variables in the algorithm. The details of the above steps are provided in the following subsections.

3.1 Transformation to Quadratic Equations

This subsection describes the first part of the proposed algorithms, which transforms an instance of the RSDP into a quadratic system. This step is essentially performed by substituting the linear equations obtained from $\mathbf{H}\mathbf{e} = \mathbf{s}$ into the quadratic equations derived from the regular structure. For efficiency, we further guess a subset of variables

and perform Gaussian elimination prior to the guessing step. The first part of the hybrid algorithm can be outlined as follows:

1. Perform Gaussian elimination on $\mathbf{H}$.
2. Guess some error-free positions of $\mathbf{e}$ and derive a new equation $\mathbf{H}'\mathbf{e}' = \mathbf{s}$ with reduced dimension.
3. Transform the matrix $\mathbf{H}'$ into row echelon form by eliminating the remaining columns.
4. Substitute the linear equations obtained from $\mathbf{H}'\mathbf{e}' = \mathbf{s}$ into the quadratic equations of the form (1).

Note that we have to repeat steps 2-4 and the second part of the algorithm multiple times, since step 2 randomly guesses a subset of the variables. The details of these steps are presented below.

Step 1: Gaussian elimination on $\mathbf{H}$.

We first transform $\mathbf{H}$ into row echelon form before guessing error-free positions. Specifically, we compute an invertible matrix $\mathbf{T}_1 \in \mathbb{F}_q^{(n-k) \times (n-k)}$ such that $\mathbf{T}_1\mathbf{H}$ takes the form

$$(\mathbf{I}_{n-k} \parallel \mathbf{Q}),$$

where $\mathbf{Q} \in \mathbb{F}_q^{(n-k) \times k}$. Then, the original equation is transformed into

$$\mathbf{T}_1\mathbf{H}\mathbf{e} = \mathbf{T}_1\mathbf{s}.$$

This reduces the complexity of transforming $\mathbf{H}'$ into row echelon form in step 3 after removing some columns from $\mathbf{H}$ in the guessing step. This improvement comes from the fact that $\mathbf{H}$ and $\mathbf{H}'$ share a large number of columns.

Step 2: Guessing of error-free positions.

We divide h blocks of $\mathbf{e}$ into the first $h - g$ and the last g blocks. This step focuses on the first $h - g$ blocks and fixes several variables in these blocks. More precisely, we divide these $h - g$ blocks into the first f blocks and the remaining $h - g - f$ blocks. We fix $u + 1$ error-free positions in each of the f blocks and u error-free positions in each of the remaining $h - g - f$ blocks. We then construct a new matrix $\mathbf{H}' \in \mathbb{F}_q^{(n-k) \times n_1}$ by removing the corresponding $(u + 1)f + u(h - g - f)$ columns from $\mathbf{H}$, where $n_1 = n - (u + 1)f - u(h - g - f)$. From the conditions for parameters (u, f, g) , we have $n_1 > n - k$. We similarly define $\mathbf{e}'$ as an n_1 -dimensional vector obtained by removing the corresponding variables from $\mathbf{e}$. Then, the linear equations to be solved are transformed into

$$\mathbf{T}_1\mathbf{H}'\mathbf{e}' = \mathbf{T}_1\mathbf{s}.$$

If the guess for these error-free positions is correct, we can recover the full vector $\mathbf{e}$ from the obtained $\mathbf{e}' = (e'_1, \dots, e'_{n_1})$ by filling in the corresponding positions with 0's. We discuss the success probability of this guessing and the number of times that we should repeat it in Subsection 4.4 below.

Step 3: Gaussian elimination on $\mathbf{T}_1\mathbf{H}'$.

After fixing u or $u + 1$ error-free positions in each of the left $h - g$ blocks in step 2, the left $(n - k)$ -by- $(n - k)$ submatrix of $\mathbf{T}_1\mathbf{H}'$ has a form in which only some columns have been eliminated. Thus, this step eliminates the remaining columns of the $(n - k)$ -by- $(n - k)$ submatrix. We compute an invertible matrix $\mathbf{T}_2 \in \mathbb{F}_q^{(n-k) \times (n-k)}$ such that $\mathbf{T}_2\mathbf{T}_1\mathbf{H}'$ is in row echelon form, and we have the equation

$$\mathbf{T}_2\mathbf{T}_1\mathbf{H}'\mathbf{e}' = \mathbf{T}_2\mathbf{T}_1\mathbf{s}. \quad (3)$$

Step 4: Substitution of linear equations.

We finally construct quadratic equations from the linear equations obtained from (3). We can express the first $n - k$ variables $e'_1, \dots, e'_{n-k}$ as linear polynomials in the remaining variables $e'_{n-k+1}, \dots, e'_{n_1}$ using (3). We here define $\mathbf{e}' = (\mathbf{e}'_1, \dots, \mathbf{e}'_h)$, where $\mathbf{e}'_i$ is obtained by removing the guessed u or $u + 1$ variables from $\mathbf{e}_i$ if $i \in [h - g]$, and $\mathbf{e}'_i = \mathbf{e}_i$ otherwise. We also define a positive integer ℓ such that

$$\sum_{i=1}^{\ell} |\mathbf{e}'_i| \leq n - k < \sum_{i=1}^{\ell+1} |\mathbf{e}'_i|.$$

Then, the first ℓ blocks $\mathbf{e}'_1, \dots, \mathbf{e}'_{\ell}$ consist of the first $n - k$ variables $e'_1, \dots, e'_{n-k}$. Further, the next block $\mathbf{e}'_{\ell+1}$ might include some of the first $n - k$ variables, and we set $\mathbf{e}'_{\ell+1} = (\mathbf{e}'_{\ell+1}^{(1)}, \mathbf{e}'_{\ell+1}^{(2)})$ such that $\sum_{i=1}^{\ell} |\mathbf{e}'_i| + |\mathbf{e}'_{\ell+1}^{(1)}| = n - k$. We then generate the quadratic equations as follows: We substitute the $n - k$ linear polynomials derived from (3) into

$$e'_i \cdot e'_j = 0 \quad (1 \leq i < j \leq n - k),$$

where e'_i and e'_j belong to the same block of $\mathbf{e}'$, and the additional case of

$$e'_i \cdot e'_j = 0 \quad (e'_i \in \mathbf{e}'_{\ell+1}^{(1)}, e'_j \in \mathbf{e}'_{\ell+1}^{(2)}).$$

The resulting system consists of the quadratic equations in the $n_1 - (n - k)$ variables $e'_{n-k+1}, \dots, e'_{n_1}$. We describe how to solve this system in the following subsections.

3.2 XFL-based Method for Solving Polynomial Systems

This subsection describes the XFL-based method for solving the quadratic system obtained in Subsection 3.1. This variant devises a way to combine guessing some variables and solving the system based on the idea of the XFL algorithm [14, 17]. Furthermore, we allow the solving degree $d \geq 2$ to be freely chosen, unlike the hybrid algorithm [13]. This XFL-based variant proceeds as follows:

1. Construct the Macaulay matrix and apply Gaussian elimination to a part of the matrix.
2. Guess some error-free positions of the last g blocks of $\mathbf{e}'$.
3. Perform Gaussian elimination on the remaining part of the Macaulay matrix.

We explain how to estimate the solving degree d in Subsection 4.1 below. The details of these steps are presented below.

Step 1: Partial elimination on the Macaulay matrix.

In this step, we first generate the Macaulay matrix at the solving degree d from the quadratic equations derived in Subsection 3.1. We denote this Macaulay matrix $\mathcal{M}$. The matrix $\mathcal{M}$ is constructed over a quotient ring defined as follows: We denote by $\mathcal{B}_1$ the set of polynomials $e'_i \cdot e'_j$ where $n - k < i < j \leq n_1$ and e'_i and e'_j belong to the same block of $\mathbf{e}'$. We then define the quotient ring Q as $\mathbb{F}_q[e'_{n-k+1}, \dots, e'_{n_1}] / \langle \mathcal{B}_1 \rangle$. By constructing the Macaulay matrix over this quotient ring, we can make the matrix size smaller and reduce the complexity of the algorithm.

We then analyze the structure of the Macaulay matrix $\mathcal{M}$. The columns of $\mathcal{M}$ correspond to monomials of degree $\leq d$ in Q . We define M_1 as the subset of these monomials only in $e'_{n-k+1}, \dots, e'_{n_1-\beta g}$, none of which contains the last βg variables to be guessed in the following step. We also define the complementary subset as M_2 . Meanwhile, the rows of $\mathcal{M}$ correspond to the polynomials obtained by multiplying monomials of degree $\leq d - 2$ in Q into the quadratic polynomials obtained in Subsection 3.1. We also define I_1 as the subset of these polynomials obtained by multiplying monomials only in $e'_{n-k+1}, \dots, e'_{n_1-\beta g}$ into the quadratic equations, and define I_2 as the subset of the remaining polynomials. Then, the Macaulay matrix $\mathcal{M}$ is divided into the four submatrices as follows:

	M_1	M_2
I_1	(a)	(b)
I_2	(c)	(d)

where we denote the upper-left, upper-right, lower-left, and lower-right submatrices as (a), (b), (c), and (d), respectively. Note that the submatrix (c) is a zero matrix, since any monomial appearing with a nonzero coefficient in a polynomial of I_2 includes at least one of the last βg variables from the definition of I_2 .

For the Macaulay matrix $\mathcal{M}$ with the above structure, we apply the following operations:

- (1) Apply Gaussian elimination on the submatrix (a).
- (2) Perform the same row operations as those of (1) on the submatrix (b).

In the first step, we eliminate the columns corresponding to M_1 . Note that these columns are unaffected by the guessing of error-free positions in the last g blocks, according to the definition of M_1 . In the second step, we perform the corresponding row operations for consistency of row transformations on the matrix.

As a result of these operations, the matrix $\mathcal{M}$ is transformed into the following form

	M_1	M_2
I_1		
	0	
I_2	0	(d)

Note that the submatrix (a) is column full-rank with high probability if the solving degree is chosen appropriately. Then, the following two steps of the algorithm can focus on the lower part of (b) and (d), since other rows are already eliminated in the above operations. This strategy reduces the computational cost for each iteration of guessing error-free positions in the last g blocks.

Step 2: Guessing variables in the g blocks.

In this step, we guess $\beta - 1$ error-free positions in each of the last g blocks. Note that all elements of the last g blocks are included in $e'_{n-k+1}, \dots, e'_{n_1}$ due to the conditions for parameters (u, f, g) . After removing the corresponding variables, the resulting system contains $n_1 - (n - k) - (\beta - 1)g$ variables. We repeat this step together with the following one until the error positions in the g blocks are correctly anticipated.

Step 3: Gaussian elimination of the remaining submatrix.

In this step, we apply Gaussian elimination on the lower-right part of $\mathcal{M}$, after removing the columns corresponding to the variables fixed to zero in step 2. If d is sufficiently large, we obtain a univariate equation, from which the solution can be easily recovered in the same way as in the original XL algorithm. Finally, we check whether the resulting solution, after filling the fixed error-free positions with zeros, satisfies both $\mathbf{H}\mathbf{e} = \mathbf{s}$ and the regular condition. If the solution is invalid, the procedure returns to the guessing steps in either the first or second part of the algorithm.

3.3 WXL-based Method for Solving Polynomial Systems

This subsection describes another variant for solving the quadratic system obtained in Subsection 3.1, which is based on the Wiedemann XL (WXL) algorithm [18]. As in the XFL-based method described in Subsection 3.2, we allow the solving degree $d \geq 2$ to be freely chosen. The second part of the algorithm proceeds as follows:

1. Guess some error-free positions in the last g blocks of $\mathbf{e}'$.
2. Solve the resulting system by applying WXL.

The details of these steps are given below: In step 1, we guess $\beta - 1$ error-free positions in each of the last g blocks as in step 2 of the XFL-based variant. In step 2, we apply WXL to the resulting system. WXL employs Wiedemann's algorithm, which efficiently computes kernel vectors of a matrix by exploiting its sparsity, instead of performing Gaussian elimination in the XL framework. This step first constructs the Macaulay matrix from the resulting equations over the quotient ring Q defined by the relations

$e'_i \cdot e'_j = 0$, as in the XFL-based method. We then apply WXL to the Macaulay matrix to derive a univariate equation. As explained in Subsection 3.2, the XFL-based method partially performs Gaussian elimination before guessing some variables. However, the WXL-based method cannot apply such partial elimination, since this preprocessing makes the remaining part of the matrix dense, which prevents WXL from working efficiently.

Remark 1 (XFL-based variant with $d = 2$ and the hybrid algorithm [13]) The proposed XFL-based variant is constructed by extending the hybrid algorithm so that the solving degree d can be chosen freely. Therefore, when $d = 2$, the XFL-based variant essentially follows the same procedure as the hybrid algorithm. One difference is that the proposed XFL-based variant constructs and processes the Macaulay matrix over the quotient ring Q defined by the regular relations, whereas the hybrid algorithm performs the corresponding procedure over the ordinary polynomial ring.

4 Complexity

This section first explains a method for estimating the solving degree determined for the second parts of the proposed algorithms in Subsections 3.2 and 3.3. We then analyze the structure of the Hilbert series used in this analysis and experimentally validate the resulting solving-degree estimates. Finally, we provide an estimation of the overall time complexity of the proposed algorithms.

4.1 Solving Degree

This subsection discusses the solving degree used in the second part of the proposed algorithms to solve the obtained quadratic systems. We estimate the solving degree by analyzing the coefficients of the Hilbert series constructed from the given equations, as in the original XL algorithm. In this subsection, we first recall the Hilbert series corresponding to the system derived from the RSDP, introduced by Briaud and Øygarden [11]. We then discuss its applicability to the proposed algorithms in Section 3. Note that the solving degree takes the same value for both the XFL-based and WXL-based variants, since the algorithmic structure does not affect it.

We first recall the Hilbert series used to estimate the solving degree of the algebraic algorithm proposed by Briaud and Øygarden. Recall that the difference between the algorithm proposed in [11] and the proposed algorithms is as follows: The algorithm in [11] solves the $n - k$ linear equations obtained from $\mathbf{H}\mathbf{e} = \mathbf{s}$ directly, without transforming them into quadratic equations. Here, for simplicity, we discuss the Hilbert series for the system in n variables without considering the guessing of a subset of the variables.

We denote by $\mathcal{P}$ the set of the $n - k$ linear polynomials given by each element of $\mathbf{H}\mathbf{e} - \mathbf{s}$. We also denote by $\mathcal{B}$ the set of quadratic polynomials $\{e_{k,i} \cdot e_{k,j}\}_{k \in [h], i \neq j}$ derived from the regular structure. In Assumption 1 in [11], it is assumed that all non-trivial relations of $\mathcal{P}^{(h)}$ in $\mathbb{F}_q[\mathbf{e}]/\langle \mathcal{B}^{(h)} \rangle$ arise naturally from the regular structure. Here, for a subset $\mathcal{F} \subseteq \mathbb{F}_q[\mathbf{e}]$, we denote by $\mathcal{F}^{(h)}$ the set of polynomials $f^{(h)}$ with

$f \in \mathcal{F}$, which consists only of the terms of f with degree $\deg f$. Under this assumption, Theorem 1 in [11] shows that the Hilbert series of $\langle (\mathcal{P} \cup \mathcal{B})^{(h)} \rangle$ is given by

$$\left[(1-z)^{n-k} \cdot \left(1 + \beta \cdot \frac{z}{1-z} \right)^h \right]_+, \quad (4)$$

where $[\cdot]_+$ means truncation after the first non-positive coefficient. In this series, the coefficient of $z^{d'}$ in $(1 + \beta z/(1-z))^h$ represents the number of monomials of degree d' in $\mathbb{F}_q[\mathbf{e}]/\langle \mathcal{B}^{(h)} \rangle$, and the term $(1-z)^{n-k}$ corresponds to the $n-k$ linear equations.

We then discuss the applicability of this estimation to the proposed algorithms. Note that our analysis is also conducted under Assumption 1 in [11]. We denote by $\mathcal{B}_1$ the set of quadratic polynomials of the form $e_i \cdot e_j$ where $n-k < i < j \leq n$ and e_i and e_j belong to the same block. We also define $\mathcal{B}_2$ as $\mathcal{B} \setminus \mathcal{B}_1$. The proposed algorithms eliminate the first $n-k$ variables using $\mathcal{P}$ while preserving the regular structure, and solve the resulting equations in the remaining k variables of $\mathbf{e}$. This corresponds to the computation in the quotient ring $\mathbb{F}_q[\mathbf{e}]/\langle \mathcal{P} \cup \mathcal{B}_1 \rangle$. Then, the proposed algorithms solve the equations obtained from $\mathcal{B}_2$ in $\mathbb{F}_q[\mathbf{e}]/\langle \mathcal{P} \cup \mathcal{B}_1 \rangle$. From these discussions, the algorithm by Briaud and Øygarden and the proposed algorithms coincide in that both find a zero of $\langle \mathcal{P} \cup \mathcal{B}_1 \cup \mathcal{B}_2 \rangle$. Therefore, we can estimate the solving degree of the proposed algorithms by using the Hilbert series given as (4).

We finally estimate the solving degree of the proposed algorithms while taking into account the subset of variables that are guessed.

Proposition 1 *The solving degree of the proposed algorithms is obtained as the smallest degree d where the coefficient of z^d becomes non-positive in the following series*

$$(1-z)^{n-k} \cdot \left(1 + (\beta - u - 1) \cdot \frac{z}{1-z} \right)^f \cdot \left(1 + (\beta - u) \cdot \frac{z}{1-z} \right)^{h-g-f} \cdot \left(\frac{1}{1-z} \right)^{g+1}. \quad (5)$$

The above series is derived from the fact that the proposed algorithms fix the values of $u+1$ variables in f blocks, u variables in $h-g-f$ blocks, and $\beta-1$ variables in g blocks, respectively. In the above series, the exponent $g+1$ of the factor $1/(1-z)$ accounts for the g variables corresponding to the last g blocks, while the single additional unit in the exponent arises from the inhomogeneous structure.

4.2 Structural Analysis of Hilbert Series

In this subsection, we discuss the applicability of the series (4) to the system solved in the proposed algorithms from the perspective of its structural form. In particular, we analyze the linear dependencies among the equations to provide a structural interpretation of the series.

For simplicity, we assume that $\beta \mid (n-k)$ and set $\alpha := (n-k)/\beta$. We reinterpret the series (4) within the framework of transforming the $n-k$ linear equations into

quadratic equations as follows:

$$((1-z)^{\beta-1} \cdot (1-z+\beta z))^{\alpha} \cdot \left(1 + \beta \cdot \frac{z}{1-z}\right)^{h-\alpha}. \quad (6)$$

In this series, the term $(1 + \beta z/(1-z))^{h-\alpha}$ corresponds to the k variables in the system solved in the proposed algorithms, which follow the regular structure with $k/\beta = h - \alpha$ blocks and block size β . We then try to show that the term

$$((1-z)^{\beta-1} \cdot (1-z+\beta z))^{\alpha},$$

represents the behavior of the quadratic equations constructed in Subsection 3.1.

We write the $n - k$ linear equations obtained from $\mathbf{H}\mathbf{e} = \mathbf{s}$ in the form

$$e_i - L_i(e_{n-k+1}, \dots, e_n) = 0 \quad (i \in [n - k]),$$

where L_i is a linear polynomial in $e_{n-k+1}, \dots, e_n$. Then, from the first β parity check equations, the proposed algorithms clearly construct $\binom{\beta}{2}$ equations of the form $L_i \cdot L_j = 0$ for any $1 \leq i < j \leq \beta$. We here focus on the relation between these $\binom{\beta}{2}$ equations and the term $(1-z)^{\beta-1} \cdot (1-z+\beta z)$ in (6). Note that this term can be expanded as

$$\begin{aligned} (1-z)^{\beta-1} \cdot (1-z+\beta z) &= \sum_{i=0}^{\beta} (-1)^{i+1} (i-1) \binom{\beta}{i} z^i \\ &= 1 - \binom{\beta}{2} z^2 + 2 \binom{\beta}{3} z^3 - 3 \binom{\beta}{4} z^4 + \dots \end{aligned} \quad (7)$$

In (7), the coefficient of z^2 reflects the presence of $\binom{\beta}{2}$ quadratic equations constructed from $L_1, \dots, L_{\beta}$. We then consider the case of equations of degree 3. By multiplying $e_{n-k+1}, \dots, e_n$ into the equations $L_i \cdot L_j = 0$, we can construct $k \binom{\beta}{2}$ equations of degree 3. However, there exist some linear relations among these equations. We here clearly have the equations $L_m \cdot (L_i \cdot L_j) = L_j \cdot (L_i \cdot L_m) = L_i \cdot (L_j \cdot L_m)$ for any $1 \leq i < j < m \leq \beta$. These equations directly indicate that we have two linear relations among the following $3k$ equations

$$\begin{aligned} e_{n-k+1} L_i \cdot L_j &= 0, \dots, e_n L_i \cdot L_j = 0, \\ e_{n-k+1} L_i \cdot L_m &= 0, \dots, e_n L_i \cdot L_m = 0, \\ e_{n-k+1} L_j \cdot L_m &= 0, \dots, e_n L_j \cdot L_m = 0. \end{aligned}$$

This implies that there are $2 \binom{\beta}{3}$ linear relations among the obtained $k \binom{\beta}{2}$ equations of degree 3. This number of relations corresponds precisely to the coefficient of z^3 in (7).

Extending this viewpoint, the coefficients in (7) can be interpreted through the relations among the equations and the higher-order relations among these relations. Indeed, this interpretation also applies to degrees higher than three, where the coefficients can be interpreted as reflecting successive dependencies among the equations

Table 1 Comparison between the theoretical estimates in (5) and experimental coranks of the Macaulay matrices in the proposed algorithms over $\mathbb{F}_{16}$. In each entry, the upper and lower values denote the theoretical and experimental coranks, respectively. We set $(u, f, g) = (0, 0, 0)$.

(n, k, β)	solving degree	$d = 2$	$d = 3$	$d = 4$	$d = 5$	$d = 6$
$(16, 5, 2)$	4	13	8	1	–	–
		13	8	1	–	–
$(15, 7, 3)$	4	21	10	1	–	–
		21	10	1	–	–
$(16, 9, 4)$	4	31	12	1	–	–
		31	12	1	–	–
$(20, 12, 5)$	4	51	15	1	–	–
		51	15	1	–	–
$(24, 8, 2)$	5	33	57	21	1	–
		33	57	21	1	–
$(24, 12, 3)$	5	67	159	96	1	–
		67	159	96	1	–
$(28, 17, 4)$	5	129	440	546	1	–
		129	440	546	1	–
$(30, 19, 5)$	5	150	460	65	1	–
		150	460	65	1	–
$(34, 12, 2)$	6	74	234	409	221	1
		74	234	409	221	1
$(36, 13, 2)$	6	87	308	643	630	1
		87	308	643	630	1

and the relations generated at lower degrees. Thus, the coefficients in (7) reflect this successive structure of dependencies.

4.3 Experimental Validation

We verify the solving-degree estimation in Subsection 4.1 by comparing the Hilbert-series prediction with experimental values computed using Magma [26] from concrete RSDP instances. Indeed, the coefficient of z^d in the series (5) is expected to equal the corank of the degree- d Macaulay matrix constructed in the proposed algorithms, and the solving degree is accordingly estimated as the smallest degree d at which this coefficient becomes non-positive. Based on this observation, we conduct experiments to examine whether the coefficients of the Hilbert series agree with the coranks of the corresponding Macaulay matrices for degrees d up to the predicted solving degree.

In Table 1, we fix $q = 16$ and $(u, f, g) = (0, 0, 0)$, and consider RSDP parameter sets whose predicted solving degrees are between 4 and 6 for computational feasibility. For each parameter set, we generate an RSDP instance with a regular solution and construct the Macaulay matrix at each degree from 2 to the predicted solving degree.

We then compute its corank and compare it with the theoretical corank obtained from the corresponding Hilbert series. We repeat this experiment ten times for each parameter set, and in all trials, the experimental coranks coincide exactly with the theoretical values at every considered degree. These results suggest that the Hilbert-series-based estimation is consistent with the observed degree behavior of the polynomial systems arising from the RSDP.

4.4 Complexity Estimation

This subsection presents the complexity estimation of the proposed algorithms. We denote by $T_{1,i}$, $T_{2,i}$, and $T_{2',i}$ the complexities of step i in the first part described in Subsection 3.1, the XFL-based second part described in Subsection 3.2, and the WXL-based second part described in Subsection 3.3, respectively. We also denote by P_1 and P_2 the success probabilities of the guessing steps of the first and second parts of the algorithm, respectively. Then, the overall complexities of the proposed algorithms with the XFL-based and WXL-based methods are given as follows:

Theorem 2 *The overall time complexities of the proposed algorithms with the XFL-based and WXL-based methods are given as*

$$P_1^{-1} \cdot (T_{1.3} + T_{2.1} + P_2^{-1} \cdot T_{2.3}), \quad (8)$$

$$P_1^{-1} \cdot (T_{1.3} + P_2^{-1} \cdot T_{2'.2}), \quad (9)$$

respectively. The estimate for each part is given as

$$\begin{aligned} P_1 &= \left(1 - \frac{u+1}{\beta}\right)^f \cdot \left(1 - \frac{u}{\beta}\right)^{h-g-f}, \\ P_2 &= \left(\frac{1}{\beta}\right)^g, \\ T_{1.3} &= (n-k) \cdot \left(n-k - \left\lfloor \frac{n-k}{\beta} \right\rfloor \cdot (\beta-u-1)\right) \cdot \left(n_1 - \left\lfloor \frac{n-k}{\beta} \right\rfloor \cdot (\beta-u-1)\right), \\ T_{2.1} &= m_{\leq d}^\omega + m_{\leq d}^2 \cdot (m'_{\leq 2} - m_{\leq 2}) + m_{\leq d} \cdot (\bar{m} - m_{\leq d}) \cdot (m'_{\leq d} - m_{\leq d}), \\ T_{2.3} &= (m''_{\leq d} - m_{\leq d})^\omega, \\ T_{2'.2} &= 3 \cdot m''_{\leq 2} \cdot m''_{\leq d}^2. \end{aligned}$$

Here, d denotes the solving degree estimated using (5). The quantities $m_{\leq d'}$, $m'_{\leq d'}$, and $m''_{\leq d'}$ denote the numbers of monomials of degree at most d' in $e'_{n-k+1}, \dots, e'_{n_1-\beta g}$, in $e'_{n-k+1}, \dots, e'_{n_1}$, and in $e'_{n-k+1}, \dots, e'_{n_1-\beta g}$ together with the g variables in the last g blocks that remain after the guessing step in the second part of the algorithms, respectively. Furthermore, $\bar{m}$ denotes the number of rows of the submatrix (a) of the Macaulay matrix, and $2 \leq \omega < 3$ denotes the exponent of matrix multiplication.

Note that, in the above estimations, we only consider the complexity of the dominant parts. In the following, we describe the estimate of the probability and complexity of each part used in the above estimation.

Success probability P_1 .

In step 2 of the first part, we guess $u + 1$ error-free positions in f blocks and u error-free positions in $h - g - f$ blocks. Then, the success probability P_1 of this guessing is estimated as

$$P_1 = \left(1 - \frac{u+1}{\beta}\right)^f \cdot \left(1 - \frac{u}{\beta}\right)^{h-g-f}.$$

This is because the success probability of guessing β' error-free positions in one block is given as $1 - \frac{\beta'}{\beta}$. In the proposed algorithms, we repeat this guessing and the subsequent steps P_1^{-1} times to obtain a solution with sufficiently high probability.

Success probability P_2 .

In both the XFL-based and WXL-based variants, we guess the error position in the last g blocks of $\mathbf{e}'$. As in the above discussion, the success probability P_2 of this step is given by

$$P_2 = \left(\frac{1}{\beta}\right)^g.$$

Similarly, this guessing and the following procedures are repeated P_2^{-1} times to achieve a sufficiently high success probability.

Complexity $T_{1.3}$.

This step eliminates the remaining part of $\mathbf{T}_1\mathbf{H}'$ that has not yet been eliminated, after guessing some elements in the first $h - g$ blocks. The number of columns already eliminated in $\mathbf{T}_1\mathbf{H}'$ is approximately $\lfloor \frac{n-k}{\beta} \rfloor \cdot (\beta - u - 1)$. Thus, this step eliminates only the remaining $n - k - \lfloor \frac{n-k}{\beta} \rfloor \cdot (\beta - u - 1)$ columns in the left $(n - k)$ -by- $(n - k)$ submatrix of $\mathbf{T}_1\mathbf{H}'$. The complexity $T_{1.3}$ of this step is thus given by

$$T_{1.3} = (n - k) \cdot \left(n - k - \lfloor \frac{n-k}{\beta} \rfloor \cdot (\beta - u - 1) \right) \cdot \left(n_1 - \lfloor \frac{n-k}{\beta} \rfloor \cdot (\beta - u - 1) \right),$$

where $n_1 = n - (u + 1)f - u(h - g - f)$ denotes the number of variables remaining after the guessing step in the first part of the algorithm in Subsection 3.1.

Complexity $T_{2.1}$.

We first estimate the size of the Macaulay matrix $\mathcal{M}$ by using the Hilbert series discussed in Subsection 4.1. We denote by $m_{\leq d'}$ the number of possible monomials in $e'_{n-k+1}, \dots, e'_{n_1-\beta g}$ of degree $\leq d'$. Similarly, we denote by $m'_{\leq d'}$ the number of possible monomials in $e'_{n-k+1}, \dots, e'_{n_1}$ of degree $\leq d'$. Recall that n_1 variables $\mathbf{e}'$ have the block structure such as $(\mathbf{e}'_1, \dots, \mathbf{e}'_h)$ as described in Subsection 3.1. Then, the first ℓ blocks $\mathbf{e}'_1, \dots, \mathbf{e}'_\ell$ and the first part $\mathbf{e}'_{\ell+1}^{(1)}$ of $\mathbf{e}'_{\ell+1}$ are eliminated through step 4 of the first part of the algorithm. Then, following the discussion in Subsection 4.1, we

can estimate $m_{\leq d'}$ and $m'_{\leq d'}$ as the coefficients of $z^{d'}$ in

$$\begin{aligned} & \left(1 + |\mathbf{e}'_{\ell+1}| \cdot \frac{z}{1-z}\right) \cdot \prod_{i=\ell+2}^{h-g} \left(1 + |\mathbf{e}'_i| \cdot \frac{z}{1-z}\right) \cdot \left(\frac{1}{1-z}\right), \\ & \left(1 + |\mathbf{e}'_{\ell+1}| \cdot \frac{z}{1-z}\right) \cdot \prod_{i=\ell+2}^h \left(1 + |\mathbf{e}'_i| \cdot \frac{z}{1-z}\right) \cdot \left(\frac{1}{1-z}\right), \end{aligned}$$

respectively. Then, from the definition, the numbers of columns of the submatrices (a) and (b) are given by $m_{\leq d}$ and $m'_{\leq d} - m_{\leq d}$, respectively. Further, we estimate the number $\bar{m}$ of rows of the submatrix (a). We here suppose that we randomly reduce some redundant rows to make the Macaulay matrix as small as possible for efficiency, as in the plain XL. Then, from the discussion in Subsection 4.1, we determine the size of $\bar{m}$ such that $\bar{m} - m_{\leq d}$ is equal to the absolute value of the coefficient of z^d in

$$(1-z)^{n-k} \cdot \left(1 + (\beta - u - 1) \cdot \frac{z}{1-z}\right)^f \cdot \left(1 + (\beta - u) \cdot \frac{z}{1-z}\right)^{h-g-f} \cdot \left(\frac{1}{1-z}\right),$$

since the submatrix (a) corresponds only to the variables in the first $h-g$ blocks of $\mathbf{e}'$.

We then estimate the complexity of step 1 in the XFL-based part of the proposed algorithm. In this step, we first perform Gaussian elimination on the upper $m_{\leq d}$ -by- $m_{\leq d}$ submatrix of (a), whose complexity is estimated as

$$m_{\leq d}^{\omega}.$$

We then perform the same row operation on the upper $m_{\leq d}$ rows of (b). From the structure of this Macaulay matrix, each row of (b) contains at most $m'_{\leq 2} - m_{\leq 2}$ nonzero elements. Thus, the complexity for this operation is computed as

$$m_{\leq d}^2 \cdot (m'_{\leq 2} - m_{\leq 2}).$$

We finally have to add these upper $m_{\leq d}$ rows of (b) to the lower $\bar{m} - m_{\leq d}$ rows of (b) corresponding to the operations that eliminate the lower $\bar{m} - m_{\leq d}$ rows of (a). This complexity is given as

$$m_{\leq d} \cdot (\bar{m} - m_{\leq d}) \cdot (m'_{\leq d} - m_{\leq d}).$$

In conclusion, the total complexity $T_{2.1}$ of this step is given by

$$T_{2.1} = m_{\leq d}^{\omega} + m_{\leq d}^2 \cdot (m'_{\leq 2} - m_{\leq 2}) + m_{\leq d} \cdot (\bar{m} - m_{\leq d}) \cdot (m'_{\leq d} - m_{\leq d}). \quad (10)$$

Complexity $T_{2.3}$.

In this step, we perform the Gaussian elimination on the remaining part of the Macaulay matrix $\mathcal{M}$. We denote by $m''_{\leq d'}$ the number of possible monomials of degree

$\leq d'$ in $n_1 - (n - k) - (\beta - 1)g$ variables obtained after the guessing process in step 2 of this XFL-based part. From the discussion in Subsection 4.1, we can estimate $m''_{\leq d'}$ as the coefficient of $z^{d'}$ in

$$\left(1 + |\mathbf{e}'_{\ell+1}| \cdot \frac{z}{1-z}\right) \cdot \prod_{i=\ell+2}^{h-g} \left(1 + |\mathbf{e}'_i| \cdot \frac{z}{1-z}\right) \cdot \left(\frac{1}{1-z}\right)^{g+1}.$$

Then, the size of this uneliminated part is given as $m''_{\leq d} - m_{\leq d}$, and thus the complexity $T_{2.3}$ is given as

$$T_{2.3} = (m''_{\leq d} - m_{\leq d})^\omega. \quad (11)$$

Complexity $T_{2'.2}$.

In the WXL-based method described in Subsection 3.3, we apply the Wiedemann algorithm to the Macaulay matrix after guessing error-free positions in the last g blocks. From the above discussion, the size of the resulting Macaulay matrix is given as $m''_{\leq d}$. In this Macaulay matrix, each row corresponds to a polynomial generated by multiplying monomials with quadratic polynomials in $n_1 - (n - k) - (\beta - 1)g$ variables. This implies that each row contains at most $m''_{\leq 2}$ nonzero coefficients. As a result, the complexity $T_{2'.2}$ is estimated as

$$T_{2'.2} = 3 \cdot m''_{\leq 2} \cdot m''_{\leq d}{}^2.$$

5 Application to Concrete Parameters

This section estimates the complexity of the proposed algorithms for specific parameters. We consider the case of $q = 2^{128}$, where the hybrid algorithm was shown to be particularly efficient in [13]. We set $\omega = 2.8$ to compute the complexity for each parameter set.

5.1 Reference Parameters

Table 2 estimates the complexity of the proposed algorithms for reference parameter sets motivated by PCG applications and considered in [9–11, 13, 20]. For the parameter (q, n, k, h) , the block size β is defined as $\lfloor n/h \rfloor$, and we suppose that the last $n - h\lfloor n/h \rfloor$ elements of $\mathbf{e}$ are always zero if n/h is not an integer. That is, given parameters (q, n, k, h) , we consider solving the RSDP with parameters $(q, h\lfloor n/h \rfloor, k + h\lfloor n/h \rfloor - n, h)$. For each parameter set, we estimate the complexity of the XFL-based and WXL-based variants of the proposed algorithms by substituting specific parameters into the estimations given in (8) and (9), respectively. In the table, the complexity is presented as the base-2 logarithm of the number of operations over $\mathbb{F}_q$ required to perform the algorithm. For each case, we take the parameter (u, f, g) that minimizes the complexity and list the optimal parameters in the table. We further compute the solving degree d using the series (5) as explained in Subsection 4.1. For comparison, we also include the best previously known complexity estimates reported in [13]. These estimates are achieved by the hybrid algorithm for most parameter sets, whereas the

Table 2 The complexity of the proposed algorithms on the RSDP with $q = 2^{128}$. The complexity is computed as the logarithm base 2 of the number of operations over $\mathbb{F}_q$ required to perform the algorithms. We choose the parameters (u, f, g) that minimize the complexity and compute the solving degree d following the discussion in Subsection 4.1.

(n, k, h)	best [13]	XFL	(u, f, g, d)	WXL	(u, f, g, d)
$(2^{22}, 64770, 4788)$	89	91	$(0, 711, 4, 2)$	101	$(0, 4237, 0, 2)$
$(2^{20}, 32771, 2467)$	114	116	$(4, 260, 4, 2)$	115	$(0, 0, 0, 3)$
$(2^{18}, 15336, 1312)$	119	117	$(1, 73, 3, 3)$	118	$(1, 678, 0, 3)$
$(2^{16}, 7391, 667)$	132	133	$(6, 614, 3, 2)$	138	$(4, 433, 0, 3)$
$(2^{14}, 3482, 338)$	133	133	$(7, 102, 3, 2)$	140	$(7, 226, 0, 2)$
$(2^{12}, 1589, 172)$	133	131	$(7, 1, 3, 2)$	138	$(8, 12, 4, 2)$
$(2^{10}, 652, 106)$	158	154	$(5, 11, 3, 2)$	161	$(5, 20, 0, 2)$
$(2^{22}, 64770, 2735)$	87	86	$(0, 0, 0, 2)$	94	$(0, 0, 0, 2)$
$(2^{20}, 32771, 1419)$	84	86	$(1, 80, 3, 2)$	94	$(2, 899, 0, 2)$
$(2^{18}, 15336, 760)$	91	92	$(5, 526, 3, 2)$	100	$(7, 39, 0, 2)$
$(2^{16}, 7391, 389)$	96	97	$(9, 126, 3, 2)$	102	$(17, 140, 3, 2)$
$(2^{14}, 3482, 198)$	94	96	$(10, 156, 3, 2)$	97	$(15, 162, 3, 2)$
$(2^{12}, 1589, 98)$	89	92	$(14, 57, 3, 2)$	92	$(14, 57, 3, 2)$
$(2^{10}, 652, 57)$	98	94	$(9, 11, 3, 2)$	96	$(10, 5, 3, 2)$

RP algorithm proposed in [10] gives the best estimates for $(n, k, h) = (2^{14}, 3482, 198)$ and $(2^{12}, 1589, 98)$.

As shown in Table 2, when comparing the two proposed variants, the XFL-based method with $d = 2$, which corresponds to the original hybrid algorithm [13], achieves the lowest complexity for most parameter sets. This result indicates that the existing hybrid algorithm already achieves sufficient efficiency for a broad range of the reference parameters. On the other hand, for $(n, k, h) = (2^{20}, 32771, 2467)$ and $(2^{18}, 15336, 1312)$, the WXL-based method with $d = 3$ and the XFL-based method with $d = 3$, respectively, achieve slightly lower complexity. These results suggest that generalizing the solving degree can provide a modest improvement for certain parameter sets. We note that, for some parameters, there is a difference of a few bits between the complexity of the hybrid algorithm reported in [13] and that of our XFL-based method with $d = 2$. This discrepancy is mainly due to differences in the complexity estimation methodology rather than differences in the underlying algorithms.

5.2 Small Block Size Cases

Tables 3 and 4 estimate the complexity of the proposed algorithms for parameters with small block size β . These parameter sets are intended to investigate the parameter-dependent behavior of the algorithms in the small-block-size setting. We fix $n = 2^{10}$ in Table 3 and $n = 2^{16}$ in Table 4, and vary k in the range $0 < k < n$. For the parameter (q, n, k, β) , we consider solving the RSDP with parameters $(q, \beta \lfloor n/\beta \rfloor, k + \beta \lfloor n/\beta \rfloor - n, \beta)$ as in Subsection 5.1. As in Table 2, we evaluate the complexities of both the XFL-based and WXL-based variants using (8) and (9), respectively, and show the corresponding optimal parameters (u, f, g) and solving degree d . We only consider the case where $((q-1) \cdot \beta)^h$ is lower than or equal to q^{n-k} , since the problem is expected to have a unique solution in this setting. The entries corresponding to parameter sets

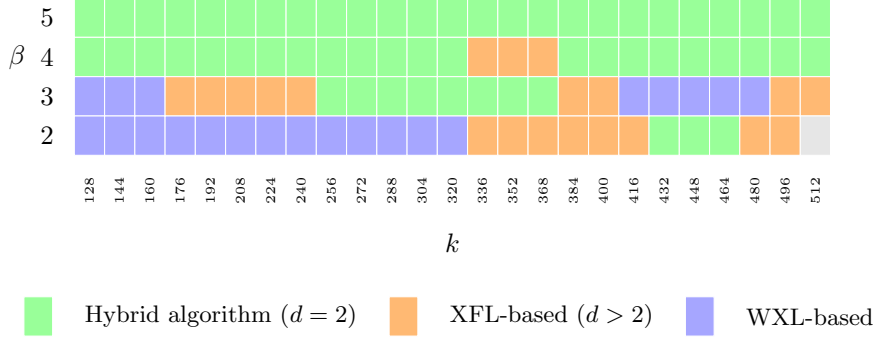


Fig. 1 Algorithm achieving the lowest estimated complexity for $q = 2^{128}$ and $n = 2^{10}$. The value of k is varied from 128 to 512 in increments of 16.

with $((q-1)\beta)^h > q^{n-k}$ are therefore left blank. For the given parameters, we compare the proposed algorithms with other algebraic algorithms for the RSDP [11–13]. The lowest complexity among these algorithms is highlighted in bold in the table for each parameter set. Note that we estimate the complexity of the hybrid algorithm [13] by using our estimation (8) with $d = 2$ for fair comparison.

As a result, Table 3 demonstrates that the proposed generalizations of the solving degree outperform the existing algorithms for some parameters with $\beta = 2, 3$. For example, the complexity of the proposed WXL-based method for parameters $(q, n, k, \beta) = (2^{128}, 2^{10}, 128, 2)$ is 101 bits, which is approximately 20 bits lower than that of the original hybrid algorithm. By contrast, we confirm that the original hybrid algorithm remains more efficient than the proposed algorithms when $\beta \geq 4$. A more detailed comparison of which algorithm achieves the lowest complexity for different parameter settings is shown in Fig. 1. Table 4 evaluates the complexity for $n = 2^{16}$ and selected block sizes in the range $5 \leq \beta \leq 20$. In particular, when k is relatively large, such as $k = 2048$, the proposed algorithms achieve the lowest complexity among the compared algorithms for all the considered values of β . The WXL-based variant achieves the lowest complexity for $\beta = 5, 10, 15$, while the XFL-based variant does so for $\beta = 20$. Compared with the results for $n = 2^{10}$ in Table 3, the proposed algorithms achieve the lowest complexity for larger values of β when $n = 2^{16}$, suggesting that their advantage extends to larger block sizes as n increases.

We consider that the reason the proposed algorithms using $d \geq 3$ are more effective in the small β case is as follows: If the block size β is small, then the constraints imposed on the variables by the regular structure are less restrictive. Thus, in the hybrid algorithm, we have to guess more variables to achieve $d = 2$ in the small β cases. The proposed algorithms relax this restriction by allowing a generalized solving degree d and thus realize the reduction of complexity. On the other hand, the results indicate that the proposed generalization of the solving degree is advantageous for the small β cases, and the original hybrid algorithm is suitable for other cases.

Remark 2 (The binary field case) This remark discusses the application of the proposed algorithms to the RSDP over the binary field $\mathbb{F}_2$. As described in [11], for the RSDP over $\mathbb{F}_2$,

Table 3 Comparison of the complexity of the proposed algorithms for the RSDP with small block sizes, where $q = 2^{128}$ and $n = 2^{10}$. The complexity is computed as the logarithm base 2 of the number of operations over $\mathbb{F}_q$ required to perform the algorithm. We choose the parameters (u, f, g) that minimize the complexity and compute the solving degree d following the discussion in Subsection 4.1.

(k, β)	AGB	Hybrid	AGB2.0	Our results			
	[11]	[13]	[12]	XFL	(u, f, g, d)	WXL	(u, f, g, d)
(128,2)	139	123	125	112	(0, 40, 6, 5)	101	(0, 6, 0, 8)
(128,3)	95	76	85	73	(0, 48, 4, 3)	73	(0, 5, 0, 5)
(128,4)	78	59	69	59	(0, 67, 4, 2)	63	(0, 5, 0, 4)
(128,5)	66	49	59	49	(0, 50, 4, 2)	55	(0, 66, 0, 2)
(256,2)	277	255	263	250	(0, 197, 5, 4)	245	(0, 144, 1, 9)
(256,3)	174	154	164	154	(0, 213, 5, 2)	157	(0, 174, 0, 4)
(256,4)	133	114	125	114	(0, 203, 4, 2)	120	(0, 216, 0, 2)
(256,5)	114	92	109	92	(0, 188, 4, 2)	98	(1, 1, 0, 2)
(384,2)	410	388	398	387	(0, 352, 4, 3)	390	(0, 336, 0, 5)
(384,3)	258	238	248	234	(0, 332, 4, 3)	238	(0, 341, 0, 3)
(384,4)	217	185	209	185	(1, 88, 4, 2)	189	(1, 47, 0, 4)
(384,5)	169	147	162	147	(1, 125, 4, 2)	153	(1, 138, 0, 2)
(512,2)	-	-	-	-	-	-	-
(512,3)	429	371	419	369	(1, 134, 4, 3)	370	(1, 94, 1, 7)
(512,4)	285	263	278	263	(1, 225, 4, 2)	269	(1, 231, 1, 2)
(512,5)	247	213	244	213	(2, 63, 4, 2)	219	(2, 27, 0, 4)

Table 4 Comparison of the complexity of the proposed algorithms for the RSDP with small block sizes, where $q = 2^{128}$ and $n = 2^{16}$. The complexity is computed as the logarithm base 2 of the number of operations over $\mathbb{F}_q$ required to perform the algorithm. We choose the parameters (u, f, g) that minimize the complexity and compute the solving degree d following the discussion in Subsection 4.1.

(k, β)	AGB	Hybrid	AGB2.0	Our results			
	[11]	[13]	[12]	XFL	(u, f, g, d)	WXL	(u, f, g, d)
(512,5)	73	48	89	48	(0, 0, 1, 2)	53	(0, 1, 0, 2)
(512,10)	73	47	89	47	(0, 0, 0, 2)	52	(0, 0, 0, 2)
(512,15)	73	48	89	48	(0, 0, 0, 2)	52	(0, 0, 0, 2)
(512,20)	73	47	89	47	(0, 0, 0, 2)	52	(0, 0, 0, 2)
(1024,5)	130	207	130	99	(0, 0, 0, 4)	91	(0, 0, 0, 4)
(1024,10)	102	84	91	77	(0, 0, 0, 3)	75	(0, 0, 0, 3)
(1024,15)	80	55	89	55	(0, 0, 5, 2)	65	(0, 68, 0, 2)
(1024,20)	74	53	89	53	(0, 0, 0, 2)	58	(0, 0, 0, 2)
(1536,5)	185	374	207	148	(0, 56, 7, 5)	130	(0, 0, 0, 6)
(1536,10)	131	163	130	105	(0, 164, 5, 3)	97	(0, 0, 0, 4)
(1536,15)	103	106	114	82	(0, 0, 0, 3)	80	(0, 0, 0, 3)
(1536,20)	103	80	89	80	(0, 337, 4, 2)	80	(0, 0, 0, 3)
(2048,5)	238	540	272	203	(0, 0, 0, 8)	168	(0, 0, 0, 8)
(2048,10)	159	242	169	134	(0, 0, 0, 5)	119	(0, 0, 0, 5)
(2048,15)	131	158	130	110	(0, 0, 0, 4)	101	(0, 0, 0, 4)
(2048,20)	112	119	126	87	(0, 35, 4, 3)	91	(0, 112, 0, 3)

one can additionally construct h linear equations of the form

$$e_{k,1} + \cdots + e_{k,\beta} = 1 \quad (k \in [h]).$$

Then, the total number of linear equations derived from the problem becomes $n - k + h$. Thus, if $n - k + h$ is greater than or equal to the number n of variables, i.e., if $h \geq k$, then the problem can be solved in polynomial time. In Subsection 5.2, we observed that the proposed algorithms are particularly efficient for small β . However, the condition $h \geq k$ holds for many parameter settings in such small β cases, and the problem can already be solved very efficiently without applying the proposed algorithms. This indicates that, in the binary field case, the generalization of the solving degree provides little additional benefit and that the hybrid algorithm with $d = 2$ remains the more appropriate choice for the binary RSDP.

6 Asymptotic Polynomial-time Regime

This section discusses the asymptotic performance of the proposed algorithms.

In [13], the authors show the parameter region in which the complexity of the hybrid algorithm is bounded by a polynomial in n as follows:

Theorem 3 (Theorem 2 in [13]) *Let $R = k/n$ and $\rho = h/n$. The total complexity T_{total} of the hybrid algorithm is at most a polynomial in n if $R^2\rho \leq n^{-1}$.*

The above bound is derived by setting the guessing parameters (u, f, g) to zero and using the solving degree $d = 2$. After eliminating the linear equations, the resulting system has $k = Rn$ variables and $O(n/\rho)$ quadratic equations, while the number of monomials of degree at most two is $O(R^2n^2)$. Comparing the numbers of equations and monomials yields the condition $R^2\rho \leq n^{-1}$ up to constant factors, which gives the bound stated in the above theorem.

We next consider the asymptotic complexity of the proposed algorithms. Since the proposed XFL-based variant includes the hybrid algorithm as the $d = 2$ case, it preserves the polynomial-time condition in Theorem 3. For an arbitrary fixed solving degree $d > 2$, the number of monomials of degree at most d is

$$O((Rn)^d),$$

while the number of equations generated by multiplying the quadratic equations by monomials of degree at most $d - 2$ is

$$O\left(\frac{n}{\rho}(Rn)^{d-2}\right).$$

In comparing these quantities, the common factor $(Rn)^{d-2}$ cancels, and hence the asymptotic polynomial-time condition remains

$$R^2\rho n = O(1)$$

for every fixed d . This condition is equivalent to the polynomial-time condition in Theorem 3.

The same observation applies to the proposed WXL-based variant. Although exploiting the sparsity of the Macaulay matrix changes the cost of the linear-algebra step, the variant is constructed from the same degree- d Macaulay matrix and therefore satisfies the same asymptotic condition for every fixed d .

Consequently, allowing a higher constant solving degree does not enlarge the polynomial-time region in terms of its asymptotic order, although its advantage appears instead for concrete parameter sets.

7 Conclusion

The regular syndrome decoding problem (RSDP) is a variant of the SDP in which the error vector is divided into equally sized blocks, each having weight exactly one. The hybrid algorithm for the RSDP, which was recently proposed by Wang et al., has been recognized as the most efficient algorithm for various parameters. In particular, the increasing diversity of RSDP-based cryptographic schemes has made it increasingly important to analyze the complexity of solving the RSDP over a wider range of parameter settings. In this work, we analyze algebraic algorithms for the RSDP constructed by refining how the XL algorithm is applied in existing algebraic approaches. In particular, we extend the hybrid algorithm with respect to the solving degree d used in the XL algorithm. The original hybrid algorithm solves the resulting quadratic system with $d = 2$, whereas the proposed algorithms solve it with an arbitrary degree d determined from the parameters. Furthermore, we provide two variants for applying the XL algorithm to the resulting system, which are based on the XFL and Wiedemann XL algorithms, respectively. As a result, we show that the proposed algorithms are particularly effective for parameter settings with small block sizes. For example, for $(q, n, k, \beta) = (2^{128}, 2^{16}, 2048, 5)$, the proposed WXL-based variant reduces the complexity by approximately 70 bits compared with the best known algebraic algorithm. On the other hand, we also confirm that, for many parameter settings with larger block sizes, the existing algorithms, particularly the hybrid algorithm, still achieve the lowest complexity. We leave the evaluation of the proposed algorithms' efficiency in terms of practical implementations as future work.

Statements and Declarations

- Funding
This work was supported by JSPS KAKENHI Grant Number JP24K20771.
- Competing Interests
The authors declare no conflict of interest.
- Code availability
The Magma code used in this work is provided in Appendices A and B.
- Data availability
The data supporting the findings of this study are included in the article.

Appendix A Magma Implementation of Proposed XFL-Based Algorithm

This appendix provides a Magma implementation of the proposed algorithm. For simplicity, we implement only the XFL-based variant, whose procedure can be described and implemented more straightforwardly than that of the WXL-based variant. The implementation is intended for functional verification rather than performance evaluation, and we have confirmed that it correctly recovers and outputs the solution for small RSDP instances. Implementing the WXL-based variant, developing optimized implementations of both the proposed and existing algorithms, and conducting a practical performance comparison among them are left for future work.

Listing 1 Code of the proposed XFL-based variant

```

1  Block := func<i,b | (i-1) div b + 1>;
2
3  RegularMonomial := function(m,blocks)
4      e:=Exponents(m);
5      for i in [1..#e] do for j in [i+1..#e] do
6          if e[i] gt 0 and e[j] gt 0 and blocks[i] eq blocks[j] then return
7              false; end if;
8          end for; end for;
9      return true;
10 end function;
11
12 ReduceRegular := function(f, blocks)
13     P := Parent(f); c,m := CoefficientsAndMonomials(f);
14     I := [i : i in [1..#m] | RegularMonomial(m[i],blocks)];
15     return (#I eq 0) select P!0 else Polynomial([c[i]:i in I],[m[i]:i in I
16         ]);
17 end function;
18
19 MonomialsLE := function(P,d,blocks)
20     L := [P];
21     for t in [0..d] do
22         for m in MonomialsOfDegree(P,t) do
23             if RegularMonomial(m,blocks) then Append(~L,m); end if;
24         end for;
25     end for;
26     return L;
27 end function;
28
29 Uses := function(m,I)
30     e:=Exponents(m);
31     for i in I do if e[i] gt 0 then return true; end if; end for;
32     return false;
33 end function;
34
35 Pivots := function(E)
36     rows:=[]; cols:=[];
37     for i in [1..Nrows(E)] do
38         J := [j : j in [1..Ncols(E)] | E[i,j] ne 0];
39         if #J gt 0 then Append(~rows,i); Append(~cols,J[1]); end if;
40     end for;
41     return rows,cols;
42 end function;
43
44 Products := function(A)

```

```

43     C := [[]];
44     for B in A do C := [x cat y : x in C, y in B]; end for;
45     return C;
46 end function;
47
48 IsRegular := function(e,beta)
49     h := #Eltseq(e) div beta;
50     return &and[#[j:j in [1..beta] | e[(i-1)*beta+j] ne 0] eq 1
51         : i in [1..h]];
52 end function;
53
54 HilbertCoefficient := function(n,k,beta,u,f,g,d)
55     h:=n div beta; R:=LazyPowerSeriesRing(RationalField(),1); z:=R.1;
56     S:=(1-z)^(n-k)
57         *(1-z+(beta-u-1)*z)^f*(1-z)^(-f)
58         *(1-z+(beta-u)*z)^(h-g-f)*(1-z)^(-(h-g-f))
59         *(1-z)^(-g-1);
60     return RationalField()!Coefficient(S,d);
61 end function;
62
63 PredictedSolvingDegree := function(n,k,beta,u,f,g : MaxDegree:=30)
64     for d in [2..MaxDegree] do
65         if HilbertCoefficient(n,k,beta,u,f,g,d) le 0 then return d; end if
66     ;
67     end for;
68     return -1;
69 end function;
70
71 GenerateRSDInstance := function(q,n,k,beta : Seed:=1)
72     assert n mod beta eq 0 and 0 lt k and k lt n;
73     SetSeed(Seed); F:=GF(q); h:=n div beta; m:=n-k;
74     E:=[F!0:i in [1..n]];
75     for i in [1..h] do
76         p:=(i-1)*beta+Random(1,beta); a:=Random(F);
77         while a eq 0 do a:=Random(F); end while;
78         E[p]:=a;
79     end for;
80     e:=Vector(F,E); Q:=Matrix(F,m,k,[Random(F):i in [1..m*k]]);
81     H:=HorizontalJoin(IdentityMatrix(F,m),Q); s:=e*Transpose(H);
82     return H,e,s;
83 end function;
84
85 FirstGuesses := function(n,beta,u,f,g)
86     h:=n div beta; A:=[];
87     for i in [1..h-g] do
88         t:=(i le f) select u+1 else u;
89         B:=[Sort(Setseq(S)):S in Subsets(Set([1..beta]),t)];
90         Append(~A,[[[i-1]*beta+j:j in J]:J in B]);
91     end for;
92     return Products(A);
93 end function;
94
95 LastGuesses := function(n,beta,g)
96     h:=n div beta;
97     return Products([[[[i-1]*beta+j]:j in [1..beta]]:i in [h-g+1..h]]);
98 end function;
99
100 BuildSystem := function(H,s,beta,zeros,g)
101     F:=BaseRing(H); m:=Nrows(H); n:=Ncols(H); h:=n div beta;
102     keep:=[i:i in [1..n]|i notin zeros]; H1:=Submatrix(H,[1..m],keep);

```

```

102   if Rank(H1) ne m then return false,0,[],[],[],[],[],[]; end if;
103   E,T:=EchelonForm(H1); pr,pc:=Pivots(E);
104   if #pc ne m then return false,0,[],[],[],[],[],[]; end if;
105   if &or[Block(keep[j],beta) gt h-g and j in pc:j in [1..#keep]] then
106     return false,0,[],[],[],[],[],[];
107   end if;
108
109   free:=[j:j in [1..#keep]|j notin pc]; P:=PolynomialRing(F,#free);
110   global:=[keep[j]:j in free]; blocks:=[Block(i,beta):i in global];
111   last:=[i:i in [1..#free]|blocks[i] gt h-g];
112   forms:=[P!0:j in [1..#keep]];
113   for i in [1..#free] do forms[free[i]]:=P.i; end for;
114   ts:=s*Transpose(T);
115   for i in [1..m] do
116     L:=P!ts[i];
117     for j in free do L-:=P!E[i,j]*forms[j]; end for;
118     forms[pc[i]]:=L;
119   end for;
120
121   eqs:=[P];
122   for b in [1..h] do
123     J:=[j:j in [1..#keep]|Block(keep[j],beta) eq b];
124     for x in [1..#J-1] do for y in [x+1..#J] do
125       a:=J[x]; c:=J[y];
126       if a in pc or c in pc then
127         q:=ReduceRegular(forms[a]*forms[c],blocks);
128         if q ne 0 then Append(~eqs,q); end if;
129       end if;
130     end for; end for;
131   end for;
132   return true,P,eqs,blocks,last,keep,forms,global;
133 end function;
134
135 Macaulay := function(P,eqs,blocks,last,d)
136   F:=BaseRing(P); C:=MonomialsLE(P,d,blocks);
137   C1:=[m:m in C|not Uses(m,last)]; C2:=[m:m in C|Uses(m,last)]; C:=C1
   cat C2;
138   index:=AssociativeArray(Integers()); base:=d+1;
139   for j in [1..#C] do
140     e:=Exponents(C[j]); index[&+[e[i]*base^(i-1):i in [1..#e]]]:=j;
141   end for;
142   R1:=[P]; R2:=[P];
143   for q in eqs do for t in MonomialsLE(P,d-2,blocks) do
144     r:=ReduceRegular(t*q,blocks);
145     if r ne 0 then
146       if Uses(t,last) then Append(~R2,r); else Append(~R1,r); end if
147     ;
148   end if;
149   end for; end for;
150   R:=R1 cat R2; M:=ZeroMatrix(F,#R,#C);
151   for i in [1..#R] do
152     a,m:=CoefficientsAndMonomials(R[i]);
153     for l in [1..#m] do
154       e:=Exponents(m[l]); M[i,index[&+[e[j]*base^(j-1):j in [1..#e
155       ]]]:=a[l];
156     end for;
157   end for;
158   return M,C1,C2,#R1,#C1;
159 end function;

```

```

159 PartialElimination := function(M,r1,c1)
160   A:=Submatrix(M,[1..r1],[1..c1]);
161   B:=Submatrix(M,[1..r1],[c1+1..Ncols(M)]);
162   D:=Submatrix(M,[r1+1..Nrows(M)],[c1+1..Ncols(M)]);
163   E,T:=EchelonForm(A); pr,pc:=Pivots(E);
164   if #pc ne c1 then return false,0,0,0; end if;
165   np:=[i:i in [1..r1]|i notin pr]; B:=T*B;
166   Ap:=Submatrix(E,pr,[1..c1]); Bp:=Submatrix(B,pr,[1..Ncols(B)]);
167   Z:=Submatrix(B,np,[1..Ncols(B)]);
168   R:=(Nrows(Z) eq 0) select D else
169     ((Nrows(D) eq 0) select Z else VerticalJoin(Z,D));
170   return true,Ap,Bp,R;
171 end function;
172
173 Position := function(S,x)
174   for i in [1..#S] do if S[i] eq x then return i; end if; end for;
175   return 0;
176 end function;
177
178 Recover := function(H,keep,forms,v)
179   F:=BaseRing(H); e:=[F!0:i in [1..Ncols(H)]];
180   for i in [1..#keep] do e[keep[i]]:=Evaluate(forms[i],v); end for;
181   return Vector(F,e);
182 end function;
183
184 DirectXL := function(H,s,beta,P,keep,forms,M,C)
185   F:=BaseRing(H); K:=NullspaceOfTranspose(M);
186   if Dimension(K) ne 1 then return false,Vector(F,[F!0:i in [1..Ncols(H)]]); end if;
187   w:=Basis(K)[1]; c:=w[Position(C,P!1)];
188   if c eq 0 then return false,Vector(F,[F!0:i in [1..Ncols(H)]]); end if;
189   ;
189   w:=c^(-1)*w; v:=[w[Position(C,P,i)]:i in [1..Rank(P)]];
190   e:=Recover(H,keep,forms,v);
191   return e*Transpose(H) eq s and IsRegular(e,beta),e;
192 end function;
193
194 SolveLastGuess := function(H,s,beta,g,P,keep,forms,global,C1,C2,Ap,Bp,R,
195   chosen)
196   F:=BaseRing(H); n:=Ncols(H); h:=n div beta;
197   Z:=[(i-1)*beta+j:i in [h-g+1..h],j in [1..beta] |
198     (i-1)*beta+j notin chosen];
199   I:=[i:i in [1..#global]|global[i] in Z];
200   J:=[j:j in [1..#C2]|not Uses(C2[j],I)];
201   if #J eq 0 then return false,Vector(F,[F!0:i in [1..n]]); end if;
202   K:=NullspaceOfTranspose(Submatrix(R,[1..Nrows(R)],J));
203   if Dimension(K) ne 1 then return false,Vector(F,[F!0:i in [1..n]]);
204   end if;
205   y:=Vector(F,#C2,[F!0:i in [1..#C2]]); z:=Basis(K)[1];
206   for i in [1..#J] do y[J[i]]:=z[i]; end for;
207   x:=(y*Transpose(Bp))*Transpose(Ap^(-1)); c:=x[Position(C1,P!1)];
208   if c eq 0 then return false,Vector(F,[F!0:i in [1..n]]); end if;
209   x:=c^(-1)*x; y:=c^(-1)*y; C:=C1 cat C2; w:=Eltseq(x) cat Eltseq(y);
210   v:=[w[Position(C,P,i)]:i in [1..Rank(P)]]; e:=Recover(H,keep,forms,v);
211   return e*Transpose(H) eq s and IsRegular(e,beta),e;
212 end function;
213
214 XFLSolve := function(H,s,beta,u,f,g,d)
215   F:=BaseRing(H); n:=Ncols(H); k:=n-Nrows(H); h:=n div beta;
216   assert n mod beta eq 0 and 0 le u and u le beta-2 and f+g le h;

```

```

215     assert (u+1)*f+u*(h-g-f)+beta*g lt k and d ge 2;
216     zero:=Vector(F,[F!0:i in [1..n]]);
217     for G1 in FirstGuesses(n,beta,u,f,g) do
218         ok,P,eqs,blocks,last,keep,forms,global:=BuildSystem(H,s,beta,G1,g)
219         ;
220         if not ok then continue; end if;
221         M,C1,C2,r1,c1:=Macaulay(P,eqs,blocks,last,d);
222         if g eq 0 then
223             ok,e:=DirectXL(H,s,beta,P,keep,forms,M,C1);
224             if ok then return true,e; end if;
225         else
226             ok,A,B,R:=PartialElimination(M,r1,c1);
227             if not ok then continue; end if;
228             for G2 in LastGuesses(n,beta,g) do
229                 ok,e:=SolveLastGuess(H,s,beta,g,P,keep,forms,global,C1,C2,
230                     A,B,R,G2);
231                 if ok then return true,e; end if;
232             end for;
233         end if;
234     end for;
235     return false,zero;
236 end function;
237
238 XFLDemo := function(q,n,k,beta,u,f,g : Seed:=1)
239     H,e,s:=GenerateRSDInstance(q,n,k,beta:Seed:=Seed);
240     d:=PredictedSolvingDegree(n,k,beta,u,f,g); assert d ne -1;
241     ok,e1:=XFLSolve(H,s,beta,u,f,g,d);
242     printf "Parameters: (q,n,k,beta)=(%o,%o,%o,%o), (u,f,g,d)=(%o,%o,%o,%o)
243         \n",
244         q,n,k,beta,u,f,g,d;
245     printf "H =\n%o\n",H;
246     printf "Planted error e = %o\n",e;
247     printf "Syndrome s = %o\n",s;
248     printf "Recovered error e_rec = %o\n",e1;
249     printf "Success = %o\n",ok;
250     return ok,H,e,s,e1,d;
251 end function;
252
253 // Example:
254 // ok,H,e,s,e1,d := XFLDemo(16,12,5,3,0,1,1 : Seed:=1);

```

Appendix B Complexity Estimation Code

This appendix provides Magma code [26] for estimating the time complexity of the proposed algorithms.

Listing 2 Code to estimate the complexity of the proposed algorithms

```

1 // Series1 returns the solving degree and \bar{m}-m_{\le d}.
2 Series1 := function(n,k,h,beta,u,f,g,MaxDegree)
3     R:=LazyPowerSeriesRing(RationalField(),1);
4     z:=R.1;
5
6     GG:=(1-z)^(n-k)
7         *((1-z+(beta-u-1)*z)*(1-z)^(-1))^f
8         *((1-z+(beta-u)*z)*(1-z)^(-1))^(h-g-f)
9         *(1-z)^(-1);
10    G:=GG*(1-z)^(-g);
11

```

```

12     d:=1;
13     repeat
14         d:=d+1;
15         until RationalField()!Coefficient(G,d) le 0 or d eq MaxDegree;
16
17     return d,Max(0,-RationalField()!Coefficient(GG,d));
18 end function;
19
20
21 // Series2 returns m_{\le d}, m_{\le 2}, m'_{\le d}, m'_{\le 2},
22 // m''_{\le d}, and m''_{\le 2}.
23 Series2 := function(h,beta,u,f,g,l,l1,d)
24     R:=LazyPowerSeriesRing(RationalField(),1);
25     z:=R.1;
26
27     G1:=((1-z+(beta-u-1)*z)*(1-z)^(-1))^(Max(f-l-1,0))
28         *((1-z+(beta-u)*z)*(1-z)^(-1))^(h-g-Max(l+1,f))
29         *(1-z)^(-1);
30
31     if l lt f then
32         G1:=G1*((1-z+(beta-u-1-l1)*z)*(1-z)^(-1));
33     else
34         G1:=G1*((1-z+(beta-u-l1)*z)*(1-z)^(-1));
35     end if;
36
37     G2:=G1*((1-z+beta*z)*(1-z)^(-1))^g;
38     G3:=G1*(1-z)^(-g);
39
40     return
41         [RationalField()!Coefficient(G1,d),
42         RationalField()!Coefficient(G1,2)],
43         [RationalField()!Coefficient(G2,d),
44         RationalField()!Coefficient(G2,2)],
45         [RationalField()!Coefficient(G3,d),
46         RationalField()!Coefficient(G3,2)];
47 end function;
48
49
50 // RSDComp returns the complexity and optimal parameters of the
51 // three variants considered in Section 5.
52 RSDComp := function(q,n,k,beta)
53     h:=Floor(n/beta);
54     k:=k+h*beta-n;
55     n:=beta*h;
56     MaxDegree:=20;
57
58     // (a) XFL-based variant with d=2
59     com2:=Log(2,(beta*(q-1))^h);
60     param2:=[];
61
62     // (b) XFL-based variant with d>2
63     comd1:=Log(2,(beta*(q-1))^h);
64     paramd1:=[];
65
66     // (c) WXL-based variant
67     comd2:=Log(2,(beta*(q-1))^h);
68     paramd2:=[];
69
70     for u in [0..beta-2] do
71         for f in [0..h] do

```

```

72     for g in [0..h-f] do
73         n1:=n-(u+1)*f-u*(h-g-f);
74
75         if n1-(n-k) gt g*beta then
76             d,M1:=Series1(n,k,h,beta,u,f,g,MaxDegree);
77
78             if d lt MaxDegree then
79                 l:=Min(Floor((n-k)/(beta-u-1)),f)
80                     +Max(Floor((n-k-(beta-u-1)*f)/(beta-u)),0);
81
82                 l1:=n-k
83                     -(beta-u-1)*Min(l,f)
84                     -(beta-u)*Max(l-f,0);
85
86                 m,m1,m2:=Series2(h,beta,u,f,g,l,l1,d);
87                 M:=m[1]+M1;
88
89                 P1:=(1-(u+1)/beta)^f
90                     *(1-u/beta)^(h-g-f);
91                 P2:=beta^(-g);
92
93                 T13:=(n-k)
94                     *(n-k-Floor((n-k)/beta)*(beta-u-1))
95                     *(n1-Floor((n-k)/beta)*(beta-u-1));
96
97                 T21:=m[1]^2.8
98                     +m[1]^2*(m1[2]-m[2])
99                     +m[1]*(M-m[1])*(m1[1]-m[1]);
100
101                 T23:=(m2[1]-m[1])^2.8;
102                 T22:=3*m2[2]*m2[1]^2;
103
104                 A1:=Log(2,
105                     P1^(-1)*(T13+T21+P2^(-1)*T23));
106
107                 A2:=Log(2,
108                     P1^(-1)*(T13+P2^(-1)*T22));
109
110                 if d eq 2 and A1 lt com2 then
111                     com2:=A1;
112                     param2:=[u,f,g,d];
113                 end if;
114
115                 if d gt 2 and A1 lt comd1 then
116                     comd1:=A1;
117                     paramd1:=[u,f,g,d];
118                 end if;
119
120                 if A2 lt comd2 then
121                     comd2:=A2;
122                     paramd2:=[u,f,g,d];
123                 end if;
124             end if;
125         end if;
126     end for;
127 end for;
128 end for;
129
130 return com2,param2,comd1,paramd1,comd2,paramd2;
131 end function;

```

References

- [1] Melchor, C.A., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Bos, J., Deneuville, J.-C., Dion, A., Gaborit, P., Lacan, J., et al.: HQC: Hamming Quasi-Cyclic Key Encapsulation. Submission to NIST Post-Quantum Cryptography Standardization Process (2022)
- [2] NIST: Post-quantum cryptography CSRC. <https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization>
- [3] Augot, D., Finiasz, M., Sendrier, N.: A family of fast syndrome based cryptographic hash functions. In: Mycrypt 2005. LNCS, vol. 3715, pp. 64–83. Springer, Berlin, Heidelberg (2005)
- [4] Hazay, C., Orsini, E., Scholl, P., Soria-Vazquez, E.: TinyKeys: A new approach to efficient multi-party computation. In: CRYPTO 2018. LNCS, vol. 10993, pp. 3–33. Springer, Cham (2018)
- [5] Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector OLE. In: CCS 2018, pp. 896–912. ACM, New York, NY, USA (2018)
- [6] Bui, D., Carozza, E., Couteau, G., Goudarzi, D., Joux, A.: Faster signatures from MPC-in-the-Head. In: ASIACRYPT 2024. LNCS, vol. 15484, pp. 396–428. Springer, Singapore (2024)
- [7] Carozza, E., Couteau, G., Joux, A.: Short signatures from regular syndrome decoding in the head. In: EUROCRYPT 2023. LNCS, vol. 14008, pp. 532–563. Springer, Cham (2023)
- [8] Cui, H., Liu, H., Yan, D., Yang, K., Yu, Y., Zhang, K.: ReSolveD: Shorter signatures from regular syndrome decoding and VOLE-in-the-Head. In: PKC 2024. LNCS, vol. 14601, pp. 229–258. Springer, Cham (2024)
- [9] Esser, A., Santini, P.: Not just regular decoding: Asymptotics and improvements of regular syndrome decoding attacks. In: CRYPTO 2024. LNCS, vol. 14925, pp. 183–217. Springer, Cham (2024)
- [10] Kim, J., Lee, C.: Reduce and Prange: Revisiting Prange’s Information Set Decoding for LPN and RSD. Cryptology ePrint Archive, Paper 2024/276 (2024). <https://eprint.iacr.org/2024/276>
- [11] Briaud, P., Øygarden, M.: A new algebraic approach to the regular syndrome decoding problem and implications for PCG constructions. In: EUROCRYPT 2023. LNCS, vol. 14008, pp. 391–422. Springer, Cham (2023)
- [12] Liu, H., Wang, X., Yang, K., Yin, L., Yu, Y.: AGB 2.0: Refined Algebraic Attack against Regular Syndrome Decoding for PCG Applications. Cryptology ePrint

- [13] Wang, T., Wang, A., Yang, K., Liu, H., Yu, Y., Zhang, J., Wang, X.: A hybrid algorithm for the regular syndrome decoding problem. In: ASIACRYPT 2025, pp. 466–497. Springer, Singapore (2025)
- [14] Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In: EUROCRYPT 2000, pp. 392–407. Springer, Berlin, Heidelberg (2000)
- [15] Bouillaguet, C., Delaplace, C., Trimoska, M.: A simple deterministic algorithm for systems of quadratic polynomials over $\mathbb{F}_2$. In: SOSA 2022, pp. 285–296. SIAM, Philadelphia, PA, USA (2022)
- [16] Joux, A., Vitse, V.: A Crossbred algorithm for solving boolean polynomial systems. In: NuTMiC 2017, pp. 3–21. Springer, Cham (2017)
- [17] Yang, B.-Y., Chen, J.-M.: All in the XL family: Theory and practice. In: ICISC 2004, pp. 67–86. Springer, Berlin, Heidelberg (2004)
- [18] Yang, B.-Y., Chen, O.C.-H., Bernstein, D.J., Chen, J.-M.: Analysis of QUAD. In: FSE 2007, pp. 290–308. Springer, Berlin, Heidelberg (2007)
- [19] Wiedemann, D.H.: Solving sparse linear equations over finite fields. IEEE Trans. Inf. Theor. **32**(1), 54–62 (1986)
- [20] Liu, H., Wang, X., Yang, K., Yu, Y.: The hardness of LPN over any integer ring and field for PCG applications. In: EUROCRYPT 2024. LNCS, vol. 14656, pp. 149–179. Springer, Cham (2024)
- [21] Faugère, J.-C., Safey El Din, M., Spaenlehauer, P.-J.: Gröbner bases of bihomogeneous ideals generated by polynomials of bidegree (1,1): Algorithms and complexity. Journal of Symbolic Computation **46**(4), 406–437 (2011)
- [22] Perlner, R.A., Smith-Tone, D.: Rainbow Band Separation is Better than we Thought. Cryptology ePrint Archive, Paper 2020/702 (2020). <https://eprint.iacr.org/2020/702>
- [23] Yang, B.-Y., Chen, J.-M.: Theoretical analysis of XL over small fields. In: ACISP 2004. LNCS, vol. 3108, pp. 277–288. Springer, Berlin, Heidelberg (2004)
- [24] Bettale, L., Faugère, J.-C., Perret, L.: Hybrid approach for solving multivariate systems over finite fields. J. Math. Cryptol. **3**, 177–197 (2009)
- [25] Yang, B.-Y., Chen, J.-M., Courtois, N.: On asymptotic security estimates in XL and Gröbner bases-related algebraic cryptanalysis. In: ICICS 2004, pp. 401–413. Springer, Berlin, Heidelberg (2004)

- [26] Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system. I. The user language. *J. Symb. Comput.* **24**(3-4), 235–265 (1997)