A Practical Optimization for Wiedemann XL

Tung Chou[1] and Ruben Niederhagen[1,2]

[1] Academia Sinica, Taiwan
blueprint@crypto.tw
[2] University of Southern Denmark, Denmark
ruben@polycephaly.org

Abstract. Wiedemann XL is a variant of the XL algorithm that has been widely used in algebraic attacks. Usually, the cost of applying Wiedemann XL is estimated as $3N^2\omega$, where N is the width of the Macaulay matrix, and ω is the average row weight of the Macaulay matrix. Among $3N^2\omega$, $2N^2\omega$ is from the 1st phase of the algorithm, while $N^2\omega$ is from the 3rd phase of the algorithm. This paper shows a practical optimization that reduces the cost of the 3rd phase by a huge factor so that its cost becomes essentially negligible compared to that of the 1st phase. Our optimization makes use of the fact that, to obtain a solution of the multivariate system, only a few coordinates of the kernel vectors are needed.

1 Introduction

XL [5] (which is an abbreviation for eXtended Linearization) is one of the most fundamental algorithms for solving multivariate polynomial systems. In XL, by multiplying the multivariate polynomials in the input system by a large set of monomials, the input system is transformed into an extended system of a larger degree and much more polynomials. Then, the extended system is considered as a large matrix, called the Macaulay matrix. A solution of the input system can be derived by applying Gaussian elimination to obtain a single-variable polynomial.

However, the Macaulay matrix is usually sparse, and given the right kernel space of the Macaulay matrix, a solution of the input system can also be found by searching through the space. The Wiedemann algorithm [11], along with its generalized version "block Wiedemann" [4] due to Coppersmith, are good candidates for finding vectors in the right kernel space of a sparse matrix. [3] thus uses (block) Wiedemann to find vectors in the right kernel space. This variant of XL is called Wiedemann XL.

Existing papers and technical reports usually estimate the computational cost of Wiedemann XL as $3N^2\omega$, where N is the width of the Macaulay matrix, and $\omega \ll N$ is the average row weight of the Macaulay matrix. Below are some examples.

– Here is a part of [7] (where "GB" stands for "Gröbner basis"):

Denote by $\mathrm{GB}(q,n,m)$ the complexity of the GB/XL algorithm for the system of m quadratic polynomials of n variables over $\mathbf{F}_q$. It is known that

$$\mathrm{GB}(q,n,m) \leq 3 \underbrace{\binom{n+d_{\mathrm{reg}}}{d_{\mathrm{reg}}}^2}_{\widehat{=}N^2} \underbrace{\binom{n+2}{2}}_{\widehat{=}w}$$

if the system is semi-regular, ...

– Here is a part of [9]:

In general $n_x \geq 2n_y$, so we easily derive a loose lower bound complexity of

$$\min_{(a,b)\prec(q,q)} 3 \underbrace{\left([t^a s^b]\mathcal{M}(t,s)\right)^2}_{\widehat{=}N^2} \underbrace{(n_x+1)(n_y+1)}_{\widehat{=}w},$$

and an upper bound complexity of

$$\min_{(a,b)\prec(q,q)} 3 \underbrace{\left([t^a s^b]\mathcal{M}(t,s)\right)^2}_{\widehat{=}N^2} \underbrace{\left(\binom{n_x+1}{2}+n_x+1\right)}_{\widehat{=}w}.$$

– Here is a part of [1]:

The cost of this algorithm on an instance with m random homogeneous equations in n variables can be estimated as the cost of

$$\underbrace{3\binom{n-1+D}{D}^2}_{\widehat{=}N^2} \underbrace{\binom{n+1}{2}}_{\widehat{=}w},$$

field multiplications, ...

– Here is a part of the round-2 specification of UOV [2]:

The estimated cost of this state-of-the-art approach is

$$\min_k \; q^k \cdot \underbrace{3\binom{n'-k+d_{n'-k,m}}{d_{n'-k,m}}^2}_{\widehat{=}N^2} \underbrace{\binom{n'-k+2}{2}}_{\widehat{=}w}(2r^2+r)$$

and is *identified* as the cost of the direct attack against UOV.

The multiplication by $2r^2+r$ turns the cost into the number of bit operations.

– Here is a part of the round-2 specification of QR-UOV [6]:

> The complexity of this approach with the Wiedemann XL (WXL) [YCBC07], which is a variant of XL, for a classical adversary is given by
>
> $$\min_k \left(O\left(q^k \cdot 3 \underbrace{\binom{d_{\deg} + n - k}{d_{\deg}}^2}_{\widehat{=} N^2} \underbrace{\binom{n-k+2}{2}}_{\widehat{=} w} \right) \right),$$
>
> where d_{reg} denotes the degree of regularity of the system.

Where does $3N^2\omega$ come from? A closer look shows that there are 3 phases in the Wiedemann algorithm. The 1st phase takes about $2N^2\omega$ field multiplications, while the 3rd phase takes about $N^2\omega$. Similarly, there are 3 phases in the block Wiedemann algorithm. If the two "block vectors" are chosen to have the same width, the 1st phase would take about $2N^2\omega$ field multiplications, and the 3rd phase would take about $N^2\omega$ (under conditions discussed in Section 2.3). As shown by Thomé [10], in either Wiedemann or block Wiedemann, the 2nd phase can be made to take $O(N^{1+\epsilon})$ field multiplications, so it seems reasonable to ignore its cost. In conclusion, although the 3rd phase is not as expensive as 1st phase, it is still a highly non-trivial part of the computation.

In fact, if the first block vector in block Wiedemann is chosen to have a larger width than the second one, the 1st phase can take much fewer field operations. For example, if the first width is chosen to be 10 times the second width, the estimated cost becomes $1.1N^2\omega$ field multiplications (under conditions discussed in Section 2.3), in which case the cost of the 3rd phase becomes rather close to that of the 1st phase.

This paper shows that the computational cost of the 3rd phase can be made asymptotically smaller than that of the 1st phase. Furthermore, with our optimization, the computational cost of the 3rd phase becomes essentially negligible compared to that of the 1st phase in practice, even when the Macaulay matrix is not super large. Our optimization makes use of the fact that, to obtain a solution of the multivariate system, only a few coordinates of the kernel vectors are needed.

We implemented the optimization using C++. We took the code for BW1 and BW2 from [3] and combined it with our code for the optimized version of BW3. The resulting code for the complete Wiedemann XL is available at:

`https://www.polycephaly.org/projects/xl/index.shtml`

As a historical note, the code, along with a brief explanation of our optimization, have been made available on the website since 2016.

2 Preliminaries

2.1 XL

Let $R = \mathbb{F}_q[x_1, \ldots, x_n]$ be a multivariate polynomial ring. Let $\mathcal{M}_d$ be the set of monomials in R of degree at most d. Given an input system of multivariate polynomials of degree $D_{\text{sys}} \geq 2$

$$\begin{aligned} f^{(1)}(x_1, \ldots, x_n) &= \sum_{\mu \in \mathcal{M}_{D_{\text{sys}}}} c_\mu^{(1)} \cdot \mu, \\ f^{(2)}(x_1, \ldots, x_n) &= \sum_{\mu \in \mathcal{M}_{D_{\text{sys}}}} c_\mu^{(2)} \cdot \mu, \\ &\vdots \\ f^{(m)}(x_1, \ldots, x_n) &= \sum_{\mu \in \mathcal{M}_{D_{\text{sys}}}} c_\mu^{(m)} \cdot \mu, \end{aligned}$$

where each $c_\mu^{(i)} \in \mathbb{F}_q$, XL aims to find a solution $(s_1, \ldots, s_n) \in \mathbb{F}_q^n$ such that

$$f^{(1)}(s_1, \ldots, s_n) = \cdots = f^{(m)}(s_1, \ldots, s_n) = 0,$$

if it exists.

Given a degree $D_{\text{xl}} \geq D_{\text{sys}}$, an extended system is formed by polynomials in

$$\left\{ \mu \cdot f \mid f \in \{f^{(1)}, \ldots, f^{(m)}\}, \mu \in \mathcal{M}_{D_{\text{xl}} - D_{\text{sys}}} \right\}.$$

The extended system is thus of degree D_{xl} and consists of

$$m|\mathcal{M}_{D_{\text{xl}} - D_{\text{sys}}}|$$

polynomials. We label these polynomials as

$$g^{(1)}, g^{(2)}, \ldots g^{(m|\mathcal{M}_{D_{\text{xl}} - D_{\text{sys}}}|)}.$$

Then the Macaulay matrix $M \in \mathbb{F}_q^{m|\mathcal{M}_{D_{\text{xl}} - D_{\text{sys}}}| \times |\mathcal{M}_{D_{\text{xl}}}|}$ is formed. M is constructed from $g^{(i)}$'s and a length-$|\mathcal{M}_{D_{\text{xl}}}|$ vector u of monomials, such that the set formed by entries of u is exactly $\mathcal{M}_{D_{\text{xl}}}$. Without loss of generality, we assume that $(u_1, \ldots, u_{n+1}) = (1, x_1, x_2, \ldots, x_n)$. M is the unique matrix such that

$$\sum_j M_{i,j} \cdot u_j = g^{(i)}$$

for each $i \in \left\{1, \ldots, m|\mathcal{M}_{D_{\text{xl}} - D_{\text{sys}}}|\right\}$.

Let $s = (s_1, \ldots, s_n)$ be a solution of the input system, and let $u(s)$ be the vector obtained by substituting $s_1, \ldots, s_n$ into u. Then $u(s)$ must be a member of the right kernel space of M. Therefore, a solution can be found by searching

through the right kernel space. To ensure that searching is feasible, usually D_{xl} is chosen such that the right kernel space is as small as possible.

The aforementioned version of XL is arguably the simplest version. There have been more variants. For example, if the input system is over $\mathbb{F}_2$, it is natural to consider the ring

$$\mathbb{F}_2[x_1, \ldots, x_n]/\langle x_1^2 - x_1, \ldots, x_n^2 - x_n \rangle.$$

Also, there have been variants of XL for specific types of systems. For example, for bilinear systems, one can perform so-called "bilinear XL" [9], where the extended system is constructed in a somewhat different way, which affects the dimensions of the Macaulay matrix. For simplicity, our discussions always assume that the simplest version is used, but the reader should be aware that our technique can also be applied to other variants of XL.

2.2 The Wiedemann Algorithm

Since the original Wiedeammn algorithm can be considered as a special case of block Wiedemann, below we simply present the block version. Block Wiedemann is parametrized by two constants $\hat{m}, \hat{n} \in \mathbb{Z}$, where $\hat{m} \geq \hat{n} > 0$. The algorithm takes as input a matrix $B \in \mathbb{F}_q^{N \times N}$ and outputs at least one (and at most $\hat{n}$) non-trivial vector in the right kernel space of B with a high probability. The original Wiedemann algorithm is simply block Wiedemann with $\hat{m} = \hat{n} = 1$.

The 1st phase. In the first phase of the algorithm, which we call BW1, two random matrices $\mathbf{x} \in \mathbb{F}_q^{N \times \hat{m}}$ and $\mathbf{z} = (z_1, \ldots, z_{\hat{n}}) \in \mathbb{F}_q^{N \times \hat{n}}$ are generated. Let $\mathbf{y} = B\mathbf{z}$. Then, matrices $a^{(i)} = \mathbf{x}^\intercal B^i \mathbf{y} \in \mathbb{F}_q^{\hat{m} \times \hat{n}}$ for $i = 0, \ldots, \frac{N}{m} + \frac{N}{n} + O(1)$ are computed.

The 2nd phase. Let $\alpha^{(i)} = (a^{(i)})^\intercal$ for all i. The second phase of the algorithm, which we call BW2, considers $\alpha^{(i)}$'s as a polynomial with matrix coefficients

$$\alpha(\lambda) = \sum_i \alpha^{(i)} \lambda^i.$$

Such a polynomial is also called a matrix polynomial. From $\alpha(\lambda)$, BW2 computes another matrix polynomial

$$\beta(\lambda) = \sum_i \beta^{(i)} \lambda^i,$$

where each $\beta^{(i)} \in \mathbb{F}_q^{(\hat{m}+\hat{n}) \times \hat{n}}$. For $\ell \in \{1, \ldots, \hat{n}\}$, let $\beta_\ell^{(i)}$ be row ℓ of $\beta^{(i)}$, and let

$$\beta_\ell(\lambda) = \sum_i \beta_\ell^{(i)} \lambda^i.$$

We also let

$$b^{(i)} = \left((\beta_1^{(i)})^\intercal \dots (\beta_{\hat{n}}^{(i)})^\intercal \right),$$

and let $b(\lambda) = \sum_i b^{(i)} \lambda^i$. In addition to $\beta(\lambda)$, BW2 also computes a "nominal degree" for each $\beta_\ell(\lambda)$, which is an upper bound on the actual degree of $\beta_\ell(\lambda)$. We denote by Δ_ℓ (resp. δ_ℓ) the nominal degree (resp. actual degree) of $\beta_\ell(\lambda)$. We also denote by $\Delta_{\max}$ (resp. $\delta_{\min}$) the maximum (resp. minimum) value among $\Delta_1, \dots, \Delta_{\hat{n}}$ (resp. $\delta_1, \dots, \delta_{\hat{n}}$).

The 3rd phase. The third phase of the algorithm, called BW3, takes $\beta_1(\lambda), \dots, \beta_{\hat{n}}(\lambda)$ as input. Let

$$\tilde{\beta}_{\ell,\nu}^{(d)}(\lambda) = \sum_{i=0}^{d} \beta_{\ell,\nu}^{(i)} \lambda^{d-i} \in \mathbb{F}_q[\lambda].$$

Define

$$w_\ell^{(d)} = \sum_{\nu=1}^{\hat{n}} \tilde{\beta}_{\ell,\nu}^{(d)}(B) z_\nu \in \mathbb{F}_q^N.$$

For each $\ell \in \{1, \dots, \hat{n}\}$, BW3 outputs $w_\ell^{(d)}$, $d \in \{\delta_\ell, \dots, \Delta_\ell\}$, if $w_\ell^{(d)} \neq 0$ and $w_\ell^{(d+1)} = 0$. Indeed, since $0 = w_\ell^{(d+1)} = B \cdot w_\ell^{(d)}$, $w_\ell^{(d)}$ is a non-trivial vector in the right kernel.

2.3 Computational cost of (block) Wiedemann

Cost of BW1. Typically, each $a^{(i)}$ is computed by first computing $B \cdot (B^{i-1}\mathbf{y})$ and then $\mathbf{x}^\intercal \cdot B^i \mathbf{y}$. The cost for multiplications between B and $B^{i-1}\mathbf{y}$ is estimated as

$$(\frac{N}{\hat{m}} + \frac{N}{\hat{n}} + O(1))(N\omega\hat{n}) \approx \frac{\hat{m}+\hat{n}}{\hat{m}} N^2\omega$$

field multiplications, where ω is the average row weight of B. In particular, when $\hat{m} = \hat{n} = 1$ as in the original Wiedemann algorithm, the cost becomes $2N^2\omega$.

The cost for multiplications between $\mathbf{x}^\intercal$ and $B^i\mathbf{y}$ is often ignored. This is probably because in practice it seems fine to let

$$\mathbf{x} = \begin{pmatrix} I_{\hat{m}} \\ 0 \end{pmatrix}.$$

In this way, $\mathbf{x}^\intercal B^i \mathbf{y}$ is simply the matrix formed by the first $\hat{m}$ rows of $B^i\mathbf{y}$. If $\mathbf{x}$ is generated as a random matrix, then each multiplication $\mathbf{x}^\intercal \cdot B^i\mathbf{y}$ takes $N\hat{m}\hat{n}$ field multiplications. In this case, as long as $\omega/\hat{m}$ is large enough, the cost for multiplications between $\mathbf{x}^\intercal$ and $B^i\mathbf{y}$ does not matter.

Cost of BW2. Coppersmith's version of BW2 takes $O(N^2)$ field operations, while Thomé's version takes $O(N^{1+\epsilon})$ field operations. We do not analyze the exact cost for BW2, since it is not so important for the purpose of this paper.

Cost of BW3. In the worst case, it is required to compute all vectors in

$$\begin{aligned} S = &\left\{ w_1^{(d)} \mid d \in \{\delta_1, \ldots, \Delta_1 + 1\} \right\} \cup \\ &\left\{ w_2^{(d)} \mid d \in \{\delta_2, \ldots, \Delta_2 + 1\} \right\} \cup \\ &\vdots \\ &\left\{ w_{\hat{n}}^{(d)} \mid d \in \{\delta_{\hat{n}}, \ldots, \Delta_{\hat{n}} + 1\} \right\}. \end{aligned}$$

Observe that

$$w_\ell^{(0)} = \mathbf{z}(\beta_\ell^{(0)})^\intercal$$

and

$$w_\ell^{(d+1)} = B \cdot w_\ell^{(d)} + \mathbf{z}(\beta_\ell^{(d+1)})^\intercal.$$

This implies that

$$\left(w_1^{(d+1)} \cdots w_{\hat{n}}^{(d+1)} \right) = B \left(w_1^{(d)} \ldots w_{\hat{n}}^{(d)} \right) + \mathbf{z} \cdot b^{(d+1)}.$$

By computing $(w_1^{(d)} \cdots w_{\hat{n}}^{(d)})$ for $d = 0, \ldots, \Delta_{\max} + 1$, all vectors in S will be computed.

According to [4], the average nominal degree is about $\frac{N}{\hat{n}}$. Therefore, if $\Delta_{\max}$ is estimated as $\frac{N}{\hat{n}}$ (as [4] also did), computing $(w_1^{(d)} \cdots w_{\hat{n}}^{(d)})$ for $d = 0, \ldots, \Delta_{\max}+1$ takes

(i) about $\frac{N}{\hat{n}}$ multiplications between B and a $N \times \hat{n}$ matrix, plus
(ii) about $\frac{N}{\hat{n}}$ multiplications between $\mathbf{z}$ and a $\hat{n} \times \hat{n}$ matrix.

Multiplications of (i) take $(\frac{N}{\hat{n}}) \cdot N\omega\hat{n} = N^2\omega$ field multiplications, while multiplications of (ii) take $(\frac{N}{\hat{n}}) \cdot N\hat{n}^2 = N^2\hat{n}$ field multiplications. As long as $\omega/\hat{n}$ is large enough, the cost for multiplications of (ii) does not matter.

As briefly mentioned in [3], a way to reduce the cost for (ii) is to make $\mathbf{z}$ a sparse matrix. If the average row weight of $\mathbf{z}$ is $\omega_{\mathbf{z}}$, the cost for (ii) becomes $N^2\omega_{\mathbf{z}}$. According to experiments we have run, it seems fine to set $\hat{n} = 128$ and let each row of $\mathbf{z}$ have exactly 16 non-zero elements.

2.4 Wiedemann XL

One might think it is straightforward to use (block) Wiedemann to find vectors in the right kernel space of a Macaulay matrix, but there is an issue regarding matrix dimensions. The (block) Wiedemann algorithm takes a square matrix as input. However, usually Macaulay matrices need to be "tall" to make the right kernel space small: They have more rows than columns. To solve the issue, in [3], some rows of the Macaulay matrix are omitted to obtain a square matrix. In the remainder of this paper, we assume that the Macaulay matrix is turned into a square matrix in this way. For ease of discussion, we also assume that

1. the shortened Macaulay matrix and the original Macaulay matrix both have nullity 1,
2. $D_{\mathrm{sys}} = 2$, and
3. $D_{\mathrm{xl}} \geq 3$.

3 An Optimized Version of BW3

3.1 Recovering the relevant part of kernel vectors

For the discussion below, we denote by $\overline{v}$ the vector formed by the first $n+1$ entries of a vector v. Similarly, we denote by $\overline{A}$ the matrix formed by the first $n+1$ rows of a matrix A. Observe that for each $w_\ell^{(d)} \in S$, the first $n+1$ entries correspond to $1, x_1, \ldots, x_n$. Therefore, instead of computing $w_\ell^{(d)}$'s in S, we propose to compute only the short vectors in

$$\begin{aligned}\overline{S} = &\left\{\overline{w_1^{(d)}} \mid d \in \{\delta_1, \ldots, \Delta_1\}\right\} \cup \\ &\left\{\overline{w_2^{(d)}} \mid d \in \{\delta_2, \ldots, \Delta_2\}\right\} \cup \\ &\vdots \\ &\left\{\overline{w_{\hat{n}}^{(d)}} \mid d \in \{\delta_{\hat{n}}, \ldots, \Delta_{\hat{n}}\}\right\}.\end{aligned}$$

To see how to compute vectors in $\overline{S}$, observe that

$$\begin{aligned} w_\ell^{(d)} &= \sum_{\nu=1}^{\hat{n}} \tilde{\beta}_{\ell,\nu}^{(d)}(B) z_\nu \\ &= \sum_{\nu=1}^{\hat{n}} \sum_{i=0}^{d} \beta_{\ell,\nu}^{(i)} B^{d-i} z_\nu \\ &= \sum_{i=0}^{d} \sum_{\nu=1}^{\hat{n}} \beta_{\ell,\nu}^{(i)} B^{d-i} z_\nu \\ &= \sum_{i=0}^{d} B^{d-i}\mathbf{z} \cdot (\beta_\ell^{(i)})^{\intercal}, \end{aligned} \tag{1}$$

which implies that

$$\overline{w_\ell^{(d)}} = \sum_{i=0}^{d} \overline{B^{d-i}\mathbf{z}} \cdot (\beta_\ell^{(i)})^{\intercal}. \tag{2}$$

q	m	n	D	N	$\hat{m}$	$\hat{n}$	BW1 $\frac{\hat{m}+\hat{n}}{\hat{m}}N^2\omega$	Our BW3 $2N(n+1)\hat{n}$
16	17	15	9	1 307 504	128	128	465 002 145 124 352	5 355 536 384
16	18	16	10	5 311 735	128	128	8 633 645 785 328 850	23 116 670 720
16	19	17	10	8 436 285	128	128	24 340 449 373 618 950	38 874 401 280
16	20	18	11	34 597 290	128	128	157 710 019 081 960 800	98 953 021 440

Table 1. Estimated costs for BW1 and the optimized version of BW3 for some parameters.

Therefore, we propose to compute each $\overline{w_\ell^{(d)}}$, $d \in \{\delta_\ell, \ldots, \Delta_\ell\}$, as a linear combination of the columns of $\overline{B^{d-i}\mathbf{z}}$'s. Interestingly, each $\overline{B^{d-i}\mathbf{z}}$ has been computed in BW1 as the first few rows of $B^{d-i}\mathbf{z}$, so there is no extra computational cost required for computing $\overline{B^{d-i}\mathbf{z}}$'s.

For each $\overline{w_\ell^{(d)}} \in \overline{S}$ we compute, we let $v = \overline{w_\ell^{(d)}}$ and consider v as a potential multiple of $(1, s_1, s_2, \ldots, s_n)$ for some solution $(s_1, s_2, \ldots, s_n)$. To check whether v is valid, we simply check if

$$f^{(1)}(\frac{v_2}{v_1}, \ldots, \frac{v_{n+1}}{v_1}) = \cdots = f^{(m)}(\frac{v_2}{v_1}, \ldots, \frac{v_{n+1}}{v_1}) = 0.$$

If so, our algorithm outputs $(\frac{v_2}{v_1}, \ldots, \frac{v_{n+1}}{v_1})$ as a solution.

3.2 Computational cost

Below we discuss the number of field multiplications required to compute vectors in $\overline{S}$. For simplicity, we ignore the cost of substituting candidate solutions into the original system.

According to Equation (2), the number of field multiplications required to compute $\overline{S}$ is

$$\begin{aligned}\sum_{\ell=1}^{\hat{n}} \sum_{d=\delta_\ell}^{\Delta_\ell} (d+1)(n+1)\hat{n} =& (n+1)\hat{n} \cdot \sum_{\ell=1}^{\hat{n}} \sum_{d=\delta_\ell}^{\Delta_\ell} (d+1) \\ \leq& (n+1)\hat{n} \cdot \sum_{\ell=1}^{\hat{n}} (\Delta_\ell - \delta_\ell + 1)(\Delta_\ell + 1)\end{aligned}$$

As analyzed in [4], the average value of the nominal degrees is about $\frac{N}{\hat{n}}$, so we simply estimate the expected value of $\Delta_\ell + 1$ as $\frac{N}{\hat{n}}$. For ease of analysis, we consider each $\beta_\ell(\lambda)$ as a random matrix polynomial of degree at most Δ_ℓ, so the expected value of $\Delta_\ell - \delta_\ell + 1$ is upper bounded by 2. Therefore, we estimate the total number of field multiplications as

$$(n+1)\hat{n}^2(\frac{2N}{\hat{n}}) = 2N(n+1)\hat{n}.$$

	$b(\lambda)$	B	$\mathbf{z}$	$\overline{B^{d-i}\mathbf{z}}$'s
Original BW3	✓	✓	✓	
Our BW3	✓			✓

Table 2. Required data for original BW3 vs. our version.

Usually $\hat{m}$ and $\hat{n}$ are considered as constants. Therefore $2N(n+1)\hat{n}$ is asymptotically smaller than the estimated cost $\frac{\hat{m}+\hat{n}}{\hat{m}}N^2\omega$ of BW1, since

$$N \geq |\mathcal{M}_2| = \omega(n).$$

In fact, the cost of BW3 is not only asymptotically smaller than that of BW1, it is actually much smaller for instances we can easily solve. We show this by comparing the values of $\frac{\hat{m}+\hat{n}}{\hat{m}}N^2\omega$ and $2N(n+1)\hat{n}$ for several parameters in Table 1.

3.3 Memory/storage cost

While our optimization is very efficient in terms of the number of field operations required, it is also important to understand how much memory/storage cost it takes. We summarize which data is required by the original BW3 and the optimized BW3 in Table 2.

We discuss below the size of each piece of data.

- $b(\lambda)$ consists of about $\frac{N}{\hat{n}}$ $\hat{n} \times \hat{n}$ matrices, so it takes $N\hat{n}$ field elements to store.

- The original Macaulay matrix can be generated solely from the input system (if u is pre-defined), which consists of only $m|\mathcal{M}_2|$ field elements. The only difference between the original Macaulay matrix and B is that B is the result of omitting some rows of the Macaulay matrix. The information of which rows are omitted can be encoded by a vector of $m|\mathcal{M}_{D_{\text{xl}}-2}|$ bits. We assume that the vector is generated from a seed of $O(1)$ bits in a pseudo-random fashion.

- $\mathbf{z}$ consists of $N\hat{n}$ field elements. We note that $\mathbf{z}$ can also be generated in a pseudo-random fashion, but how $\mathbf{z}$ is generated does not affect the conclusion of our analysis.

- Each $\overline{B^{d-i}\mathbf{z}}$ is a $(n+1) \times \hat{n}$ matrix, and there are about $\frac{N}{\hat{n}}$ matrices, so $\overline{B^{d-i}\mathbf{z}}$'s take $N(n+1)$ field elements to store.

Under the assumptions mentioned in Section 2.4 and $m = O(n)$, the data required by the original BW3 consists of

$$N\hat{n} + m|\mathcal{M}_2| + O(1) + N\hat{n} = O(N)$$

field elements, while the data required by the optimized BW3 consists of

$$N\hat{n} + N(n+1) = O(Nn)$$

field elements. Therefore, our optimization can be considered as a tradeoff between time and memory/storage.

3.4 Application to finding kernel vectors

In theory, the way we compute $\overline{w_\ell^{(d)}}$'s can be applied to computation of $w_\ell^{(d)}$'s. Indeed, Equation (1) shows that

$$w_\ell^{(d)} = \sum_{i=0}^{d} B^{d-i}\mathbf{z} \cdot (\beta_\ell^{(i)})^\intercal,$$

which means that each $w_\ell^{(d)}$ can be computed as a linear combination of columns of $B^{d-i}\mathbf{z}$'s. Following the analysis in Section 3.2, computing all $w_\ell^{(d)}$'s in S in this way takes about

$$2N(n+1)\hat{n} \cdot \frac{N}{n+1} = 2N^2\hat{n}$$

field multiplications, which is asymptotically fewer than $\frac{\hat{m}+\hat{n}}{\hat{m}} N^2\omega$ field multiplications, the estimated cost of BW1.

The main issue of this approach is that $B^{d-i}\mathbf{z}$'s take a lot of memory/storage space to store. Indeed, there are about $N/\hat{n}$ $N \times \hat{n}$ matrices to store, so N^2 field elements in total. In practice, this is quite prohibitive. Take the $N \times N$ binary matrix for factoring the 768-bit RSA modulus [8] for example. Here, $N = 192\,796\,550$, so $N^2 \approx 2^{55}$, which implies that $B^{d-i}\mathbf{z}$'s would take about 2^{52} bytes to store.

4 Implementation and Experiment Result

4.1 Implementation

From Equation (2), we have

$$\begin{aligned}(\overline{w_1^{(d)}}, \ldots, \overline{w_{\hat{n}}^{(d)}}) &= \left(\sum_{i=0}^{d} \overline{B^{d-i}\mathbf{z}} \cdot (\beta_1^{(i)})^\intercal, \ldots, \sum_{i=0}^{d} \overline{B^{d-i}\mathbf{z}} \cdot (\beta_{\hat{n}}^{(i)})^\intercal\right) \\ &= \sum_{i=0}^{d} \overline{B^{d-i}\mathbf{z}} \cdot b^{(i)}.\end{aligned}$$

Our implementation simply computes the matrix $(\overline{w_1^{(d)}}, \ldots, \overline{w_{\hat{n}}^{(d)}})$ for $d = \delta_{\min}, \ldots, \Delta_{\max}$. For each d, the matrix is computed as the sum of products between $\overline{B^{d-i}\mathbf{z}}$ and $b^{(i)}$. For each $\overline{w_\ell^{(d)}}$ we compute, if $\delta_\ell \le d \le \Delta_\ell$, we substitute the

q	m	n	D	N	$\hat{m}$	$\hat{n}$	BW1 (s)	BW2 (s)	Our BW3 (s)
16	17	15	9	1 307 504	128	128	503	329	0.6
16	18	16	10	5 311 735	128	128	7894	4546	3.3
16	19	17	10	8 436 285	128	128	20 031	11 230	5.2
16	20	18	11	34 597 290	128	128	276 775	152 166	19.2

Table 3. Timings (in seconds) for solving several instances.

candidate solution into the original system as described in Section 3.1. If a solution is found, we output it and terminate the process.

The way we implement the optimization guarantees that all vectors in $\overline{S}$ will be computed in the worst case. However, we might also compute some $\overline{w_\ell^{(d)}}$'s that are not necessary. To be precise, if $d \in \{\delta_{\min}, \ldots, \Delta_{\max}\} \setminus \{\delta_\ell, \ldots, \Delta_\ell\}$, $\overline{w_\ell^{(d)}}$ will still be computed in our implementation, but it is of no use.

4.2 Experiment result

To demonstrate the power of our BW3 optimization, we ran a few experiments on AMD EPYC 7501 32-Core CPUs. For these experiments, BW1 and BW2 are carried out using 64 cores (with 1 thread per core), while BW3 is carried out using only 1 core. The experiment result is summarized in Table 3. As shown in the table, even though BW3 is not parallelized, its running time is still essentially negligible compared to that of BW1.

It might seem that BW2 is not much faster than BW1. We note that for BW2 we simply use Coppersmith's version. As shown in [3] and [8], for reasonably large instances, using Thomé's version can make BW2 much faster than BW1.

5 Acknowledgement

Both authors are supported by the Academia Sinica Grand Challenge Program AS-GCP-114-M01. This project was partially funded by the Taiwanese National Science and Technology Council via grant NSTC 113-2221-E-001-024-MY3.

References

1. Ward Beullens. Breaking rainbow takes a weekend on a laptop. In *Annual International Cryptology Conference*, pages 464–479. Springer, 2022.
2. Ward Beullens, Ming-Shing Chen, Jintai Ding, Boru Gong, Matthias J. Kannwischer, Jacques Patarin, Bo-Yuan Peng, Dieter Schmidt, Cheng-Jhih Shih, Chengdong Tao, and Bo-Yin Yang. UOV: unbalanced oil and vinegar-algorithm specifications and supporting documentation-version 2.0, 2025. `https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-2/spec-files/uov-spec-round2-web.pdf`.

3. Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, and Bo-Yin Yang. Solving quadratic equations with XL on parallel architectures. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 356–373. Springer, 2012.
4. Don Coppersmith. Solving homogeneous linear equations over GF(2) via block wiedemann algorithm. *Mathematics of Computation*, 62(205):333–350, 1994.
5. Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 392–407. Springer, 2000.
6. Hiroki Furue, Yasuhiko Ikematsu, Fumitaka Hoshino, Tsuyoshi Takagi, Haruhisa Kosuge, Kimihiro Yamakoshi, Rika Akiyama, Satoshi Nakamura, Shingo Orihara, and Koha Kinjo. QR-UOV, 2025. `https://info.isl.ntt.co.jp/crypt/qruov/files/qruov_Spec-v2.0.pdf`.
7. Yasufumi Hashimoto. An improvement of algorithms to solve under-defined systems of multivariate quadratic equations. Cryptology ePrint Archive, Paper 2021/1045, 2021. `https://eprint.iacr.org/2021/1045`.
8. Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K Lenstra, Emmanuel Thomé, Joppe W Bos, Pierrick Gaudry, Alexander Kruppa, Peter L Montgomery, Dag Arne Osvik, et al. Factorization of a 768-bit rsa modulus. In *Annual Cryptology Conference*, pages 333–350. Springer, 2010.
9. Ray Perlner and Daniel Smith-Tone. Rainbow band separation is better than we thought. Cryptology ePrint Archive, Paper 2020/702, 2020. `https://eprint.iacr.org/2020/702`.
10. Emmanuel Thomé. Subquadratic computation of vector generating polynomials and improvement of the block wiedemann algorithm. *Journal of symbolic computation*, 33(5):757–775, 2002.
11. Douglas Wiedemann. Solving sparse linear equations over finite fields. *IEEE transactions on information theory*, 32(1):54–62, 1986.