Concrete Bit-Operation Cost of XL

For Solving Multivariate Quadratic Systems Using Wiedemann and Berlekamp–Massey

Hülya Evkan[1] and Ruben Niederhagen[2]

[1] Fraunhofer SIT, Germany, and
Radboud University, the Netherlands
hevkan@gmail.com
[2] Academia Sinica, Taiwan, and
University of Southern Denmark, Denmark
ruben@polycephaly.org

Abstract. We present a concrete bit-operation cost model for solving multivariate quadratic systems with XL using Wiedemann linear algebra and Berlekamp–Massey sequence recovery. Following the CryptAttackTester methodology, we implement XL in a circuit-oriented model and derive closed-form cost formulas for the XL, Wiedemann, and Berlekamp–Massey steps. We instantiate the model for GF(2), GF(31), and GF(256), including baseline, constant-coefficient, and bucketed matrix-evaluation variants. Experiments on small parameter sizes show that the formulas accurately predict the circuit costs, while asymptotic analysis confirms convergence to the expected leading constant factors determined by the underlying field arithmetic. We apply the resulting estimates to Fukuoka MQ Challenge instances and to multivariate candidates from the NIST additional-signature process, providing a unified bit-operation comparison of Wiedemann–XL costs across several MQ-based schemes.

Keywords: XL · multivariate quadratic systems · algebraic cryptanalysis · Wiedemann algorithm · Berlekamp–Massey

1 Introduction

Concrete security estimates for cryptographic schemes are based on the computational cost of their best known attacks. Asymptotic complexity provides a widely used and effective baseline for this purpose, and conservative parameter choices based on such analyses are generally desirable. However, if attack costs are estimated too coarsely, this can lead to overly optimistic or overly pessimistic parameter choices. In particular, overly conservative estimates caused by hidden constant factors in asymptotic cost formulas may lead to larger security parameters than necessary, incurring overhead in runtime, key sizes, or communication cost for a security margin that may exceed the intended target.

A related issue arises when comparing the security of schemes based on the same underlying hard problem but analyzed using different concrete cost models. Even when the same class of attacks is considered, differences in accounting

for field arithmetic, hidden constant factors, implementation assumptions, or optimization strategies can lead to substantially different concrete security estimates. As a result, schemes with comparable asymptotic attack complexity may nevertheless receive different concrete security estimates depending on the chosen cost model. This complicates the consistent interpretation and comparison of concrete security claims across schemes.

The CryptAttackTester (CAT) framework [BC24] by Chou and Bernstein addresses these issues by expressing cryptanalytic algorithms in a unified bit-operation model. In this framework, attack algorithms, cost predictions, and probability predictions are specified with respect to the same model of computation, and the predictions are tested by simulations on small instances. This gives concrete meaning to attack-cost estimates and reduces the risk of hidden mismatches between algorithms and their analyses.

Like any computational cost model, CAT captures some aspects of implementation while abstracting away others. Other concrete cost models are available: for example, alternative analyses may instead count field operations, machine-word operations, or incorporate explicit costs for memory accesses and other architecture-dependent effects. In particular, the cost assigned to a memory access may itself vary depending on the assumed model, for example being treated as constant, logarithmic, square-root, or linear in the size of the memory. We do not attempt to compare such models here, as analyses in these alternative frameworks provide complementary perspectives.

The initial CAT paper applies its approach to brute-force key search and information-set decoding. Here, we extend this methodology to algebraic attacks based on the eXtended Linearization (XL) algorithm. XL is a classical method for solving systems of multivariate polynomial equations over finite fields and plays a central role in algebraic cryptanalysis, including the analysis of multivariate cryptographic schemes. It is one of several approaches to this problem, alongside Gröbner basis methods such as F4 [Fau99] and F5 [Fau02], fast exhaustive search (FES) [BCC+10], and hybrid approaches such as the crossbred algorithm of Joux and Vitse [JV17], as well as various refinements of XL. Existing cost analyses of XL are typically asymptotic or based on high-level field operation counts, and they do not always account for all algorithm- or implementation-level optimizations or concrete bit-operation costs for field operations.

We address this issue by implementing XL within the CAT framework and deriving concrete cost formulas. The dominant step in XL is the solution of large, sparse linear systems. We therefore focus on a standard approach for this task, namely Wiedemann's algorithm [Wie86] using the Berlekamp–Massey algorithm [Ber68,Mas69], and analyze their cost in terms of bit operations within the CAT model.

Contributions. Our contributions are as follows:

- We provide an implementation of XL suitable for concrete cost analysis in the CAT framework.
- We derive detailed concrete cost formulas for XL using Wiedemann and Berlekamp–Massey.

– We use these estimates to compare several NIST PQC signature submissions within our unified bit-operation model.

Our results include bit-operation-level cost formulas and corresponding circuit-level implementations for the fields GF(2), GF(31), and GF(256), following the parameters of the Fukuoka MQ Challenges[3]. In addition, we provide field-operation-level cost formulas as a general abstraction, which can be instantiated for other fields by supplying a corresponding field implementation in terms of bit operations. We make our source code available as open source for verification and reproducibility:

`https://github.com/IIS-ACE-lab/xl-ops`

Structure of this paper. Section 2 recalls the XL algorithm, Wiedemann linear algebra, Berlekamp–Massey sequence recovery, and the CryptAttackTester cost model. Section 3 describes our circuit-level implementation and the baseline, const, and const+bucket variants. Section 4 derives the corresponding field-operation and bit-operation cost formulas. Section 5 validates the formulas experimentally, analyzes their asymptotic behavior, and applies the cost model to Fukuoka MQ Challenge instances and selected NIST additional-signature candidates. Section 6 concludes and outlines directions for future work.

2 Preliminaries

This section provides an overview on XL and the CryptAttackTester framework.

2.1 XL, Wiedemann, and Berlekamp–Massey

The XL algorithm is the main algebraic attack we consider in this paper. It is a method for solving systems of polynomial equations by linearization. It was introduced by Courtois, Klimov, Patarin, and Shamir in 2000 [CKPS00] and has since been a fundamental tool in the cryptanalysis of multivariate schemes. The core idea of XL is to generate additional polynomial equations by multiplying the input equations with monomials up to a chosen degree, and then to treat the resulting system as a linear system in the monomial basis.

Let $R = K[x_1, \ldots, x_n]$ be a polynomial ring over a field K, and let

$$F = \{f_1, \ldots, f_m\} \subseteq R$$

be a system of polynomial equations. XL is applied at an operating degree D. Let u denote a monomial in R. For each polynomial f_i, all monomial multiples uf_i satisfying

$$\deg(uf_i) \leq D$$

are added to the system. Equivalently, u ranges over monomials that satisfy

$$\deg(u) \leq D - \deg(f_i).$$

Algorithm 1 summarizes this procedure.

[3] `https://www.mqchallenge.org/`

Algorithm 1 XL at Degree D

Require: Polynomials $f_1, \ldots, f_m \in \mathbb{K}[x_1, \ldots, x_n]$
Ensure: A solution of the polynomial system, if found
1: Choose a degree bound D
2: Initialize $\mathcal{E} \leftarrow \emptyset$
3: **for** $i = 1$ to m **do**
4: **for all** monomials u such that $\deg(uf_i) \leq D$ **do**
5: $\mathcal{E} \leftarrow \mathcal{E} \cup \{uf_i\}$
6: Expand the equations in $\mathcal{E}$ in the monomial basis of degree at most D
7: Form the Macaulay matrix A_D
8: Solve the induced linear system in monomials
9: Recover the degree-one monomials, if possible
10: Verify the candidate solution in the original system

The choice of D determines both the size of the Macaulay matrix and the number of relations available after linearization. If D is too small, the linearized system usually does not contain enough independent relations; if it is too large, the linear algebra becomes unnecessarily expensive.

In the rest of this paper, we apply XL to multivariate quadratic systems, so each input polynomial satisfies $\deg(f_i) = 2$. Yang and Chen [YC04a] provide the minimum D required for the reliable termination of XL as

$$D_0 = \min\left\{d : [t^d]\left((1-t)^{m-n-1}(1+t)^m\right) \leq d\right\},$$

where $[t^d]p$ denotes the coefficient of t^d in p. We therefore choose the operating degree D as $D = D_0$.

For the square-free case over GF(2), the field equations $x_i^2 = x_i$ change the predicted operating degree. Following the small-field analysis of Yang and Chen [YC04b], we use the corresponding square-free estimate

$$D_0^{(2)} = \min\left\{d : [t^d]\left(\frac{(1+t)^n}{(1-t)(1+t^2)^m}\right) \leq d\right\}.$$

Once the operating degree D is fixed, the XL equations are expanded in the monomial basis of degree at most D. Each multiple uf_i with $\deg(uf_i) \leq D$ contributes one row to the Macaulay matrix A_D, whose columns are indexed by the nonconstant monomials of degree at most D; the constant terms are moved to the right-hand side. Treating the remaining monomials as independent linear variables gives a linearized system

$$A_D y = b.$$

Solving this linear system and recovering the degree-one monomials yields a candidate solution, which is then checked against the original polynomial system.

Wiedemann's algorithm is a classical method for solving large sparse linear systems over finite fields without doing Gaussian elimination [Wie86]. It is a black-box method that accesses the matrix only through matrix–vector multiplications,

Algorithm 2 Wiedemann for $By = b$

Require: Sparse matrix $B \in \mathbb{K}^{N \times N}$, right-hand side $b \in \mathbb{K}^N$
Ensure: A scalar recurrence sequence and a candidate reconstruction relation
1: Choose a random projection vector $u \in \mathbb{K}^N$
2: Set $v_0 \leftarrow b$
3: **for** $k = 0$ to $2N - 1$ **do**
4: $\quad s_k \leftarrow u^\top v_k$
5: $\quad v_{k+1} \leftarrow Bv_k$
6: Recover the minimal recurrence of $(s_0, \ldots, s_{2N-1})$
7: Reconstruct a candidate solution from the recurrence coefficients

which preserves sparsity and allows for efficient computation, which in turn leads to significant cost savings compared to applying dense methods.

Let the linear system be

$$By = b,$$

where $B \in \mathbb{F}_q^{N \times N}$. Wiedemann starts from the right-hand side b and forms the Krylov sequence

$$b,\ Bb,\ B^2b,\ \ldots.$$

Algorithm 2 summarizes this sequence-generation and reconstruction procedure.

Note that, in the context of XL, the input matrix is a Macaulay matrix, which is generally rectangular rather than square: it typically has more rows than columns. To obtain a square matrix $B \in \mathbb{F}_q^{N \times N}$, the standard approach is to randomly drop rows from the Macaulay matrix until an $N \times N$ submatrix remains.

Instead of storing the Krylov vectors, Wiedemann projects them to a scalar sequence. Choose a random vector $u \in \mathbb{F}_q^N$ and define

$$s_k = u^\top B^k b, \qquad k = 0, 1, 2, \ldots.$$

The sequence $s_0, s_1, s_2, \ldots$ satisfies a linear recurrence. Suppose that we find a nonzero polynomial

$$\mu(t) = \mu_0 + \mu_1 t + \mu_2 t^2 + \cdots + \mu_d t^d$$

such that

$$\mu_0 s_k + \mu_1 s_{k+1} + \cdots + \mu_d s_{k+d} = 0$$

for all relevant k. With high probability over the choice of u, this scalar recurrence corresponds to a polynomial relation

$$\mu(B)b = 0.$$

Writing this relation out gives:

$$\begin{aligned}\mu(B)b &= 0\\ (\mu_d B^d + \mu_{d-1}B^{d-1} + \cdots + \mu_1 B + \mu_0 I)b &= 0\\ \mu_d B^d b + \mu_{d-1}B^{d-1}b + \cdots + \mu_1 Bb + \mu_0 b &= 0\\ \mu_d B^d b + \mu_{d-1}B^{d-1}b + \cdots + \mu_1 Bb &= -\mu_0 b.\end{aligned}$$

If $\mu_0 \neq 0$, this gives

$$B\left(-\mu_0^{-1}\left(\mu_d B^{d-1}b + \mu_{d-1}B^{d-2}b + \cdots + \mu_1 b\right)\right) = b.$$

Hence a candidate solution for $By = b$ is

$$y = -\frac{\mu_d}{\mu_0}B^{d-1}b - \frac{\mu_{d-1}}{\mu_0}B^{d-2}b - \cdots - \frac{\mu_1}{\mu_0}b.$$

The scalar sequence must be long enough to recover the corresponding recurrence; in the standard Wiedemann setting, one computes $T \approx 2N$ sequence terms for an $N \times N$ matrix. Each term requires one sparse matrix-vector multiplication by B. If w denotes the average row weight of B, the sequence-generation cost is therefore $\mathrm{O}(N^2 w)$ field operations.

The Berlekamp–Massey algorithm finds the shortest linear recurrence satisfied by a given sequence. Equivalently, it computes a connection polynomial

$$\mu(x) = \mu_0 + \mu_1 x + \cdots + \mu_t x^t$$

where coefficients beyond the current linear complexity may be zero, such that the corresponding recurrence annihilates the sequence. The Berlekamp–Massey variant shown in Algorithm 3 processes the sequence term by term. At position i, using the current connection polynomial μ, it computes the discrepancy

$$d_i = \sum_{j=0}^{\min(i,t)} \mu_j s_{i-j}.$$

A zero discrepancy means that the current connection polynomial is consistent with the sequence at position i. A nonzero discrepancy indicates that the connection polynomial must be corrected.

In the classical Berlekamp–Massey update, a nonzero discrepancy usually requires division by a previous discrepancy. This introduces a finite-field inversion. To avoid this inversion, the inverse-free version (see, e.g., [CS99]) performs the update without this division. Instead of normalizing each correction, it carries the scale through the connection-polynomial update. The correction is thus written as

$$\psi(x) = \delta\mu(x) - d_i\beta(x).$$

Here $\beta(x)$ is the auxiliary polynomial used for the correction, and δ is the scale value that replaces the division by a previous discrepancy. If $d_i = 0$, this update

Algorithm 3 Inverse-free Berlekamp–Massey

Require: A sequence $s = (s_0, s_1, \ldots, s_{2t-1})$ over a field $\mathbb{F}$
Ensure: A normalized connection polynomial $\mu(x)$ with $\mu_0 = 1$, if found

$t \leftarrow \lfloor |s|/2 \rfloor$
$\mu(x) \leftarrow 1$
$\beta(x) \leftarrow x$
$\delta \leftarrow 1$
$\ell \leftarrow 0$
for $i = 0, 1, \ldots, 2t-1$ **do**
    $d_i \leftarrow \sum_{j=0}^{\min(i,t)} \mu_j s_{i-j}$ ▷ discrepancy
    $\psi(x) \leftarrow \delta\mu(x) - d_i\beta(x)$
    if $d_i = 0$ **or** $i < 2\ell$ **then**
        $\beta(x) \leftarrow x\beta(x)$
    else
        $\beta(x) \leftarrow x\mu(x)$
        $\ell \leftarrow i - \ell + 1$
        $\delta \leftarrow d_i$
    $\mu(x) \leftarrow \psi(x)$
if $\mu_0 = 0$ **then**
    return failure
else
    return $\mu_0^{-1}\mu(x)$

only rescales $\mu(x)$, so the recurrence itself does not change. If $d_i \neq 0$, the term $d_i\beta(x)$ changes the connection polynomial.

The algorithm also maintains the current linear complexity ℓ. If $d_i = 0$, or if $i < 2\ell$, the auxiliary polynomial is shifted. Otherwise, the current connection polynomial becomes the new auxiliary polynomial, the linear complexity is updated to $\ell \leftarrow i - \ell + 1$, and the current discrepancy becomes the new scale value.

After this, $\mu(x)$ is replaced by $\psi(x)$ and the algorithm continues with the next sequence term. After the last iteration, if $\mu_0 \neq 0$, the polynomial is normalized by multiplying it with μ_0^{-1}. Hence, the inverse-free update avoids inversions during the recurrence computation and requires only one finite-field inversion for the final normalization.

For an input sequence of length T, Berlekamp–Massey has quadratic complexity $\mathrm{O}(T^2)$. In the Wiedemann setting, one usually takes $T \approx 2N$, so the cost for Berlekamp–Massey is of order $\mathrm{O}(N^2)$.

Hybrid XL combines guessing variables with the XL algorithm. One first fixes some variables to field elements and then applies XL to the reduced system in the remaining variables. Let g be the total number of fixed variables and let $n' = n - g$ be the number of variables after specialization.

For underdetermined systems with $n > m$, the first $n - m$ specializations bring the system to the square regime $n' = m$. Since such systems are expected to have many solutions, this initial reduction does not add an additional exhaustive-

search penalty — with high probability, a random assignment of the first $n - m$ variables can be extended to a solution. Additional specializations beyond this point encounter a penalty: if $g = n - m + k$, then the total cost is multiplied by q^k since up to q^k guesses are required to find a solution.

In the XL–Wiedemann algorithm, the reduced system must provide enough rows to form the square linear system used in the Wiedemann step. We therefore apply XL only after reaching the overdefined regime $m \geq n' + 2$, which requires at least $k = 2$ additional guesses in the underdetermined case. Larger values of k can nevertheless be beneficial, both for initially underdetermined and already overdefined systems, when the smaller value of n' lowers the operating degree D enough to offset the additional factor q^k.

Block Wiedemann, block Berlekamp–Massey and Thomé's optimizations. Coppersmith's *block Wiedemann* algorithm [Cop94] replaces the scalar projection in Wiedemann's algorithm with a block of projections in parallel. This increases the cost per iteration in the linear-algebra step proportionally to the block size, since the matrix-vector multiplication is replaced by a matrix-matrix multiplication, but the number of iterations decreases by the same factor such that the leading-order asymptotic scaling of the linear-algebra step remains unchanged.

The block version of Berlekamp–Massey, however, encounters a linear cost penalty in the block size, which limits the maximum block size in practice. Hence, Thomé [Tho02] proposed an alternative for block Berlekamp–Massey that constructs the annihilating polynomial using a binary product tree. With asymptotically efficient multiplication algorithms, this reduces the asymptotic complexity from $\mathrm{O}(N^2)$ to $\mathrm{O}(N \log N)$, which ameliorates the cost penalty introduced by blocking.

Overall, block Wiedemann combined with Thomé's optimizations preserves the same leading-order asymptotic scaling behavior as plain Wiedemann. Its primary advantages are improved parallelization and higher success probability by producing multiple candidate solutions in parallel. We restrict the following discussion to plain Wiedemann and Berlekamp–Massey instead of their block variants.

2.2 Bit-Operation Cost Model – CryptAttackTester

Asymptotic cost formulas provide a useful baseline for estimating the computational cost of cryptanalytic algorithms. This can give us informed bit-security levels — but it often is not sufficient for directly comparing the cost of different attack types and hence the concrete security of cryptographic schemes. Furthermore, asymptotic cost formulas might make an attack appear significantly cheaper than it is, leading to overly conservative parameter choices and therefore lower performance or higher implementation cost. Finally, there is a tension between logical operations and memory consumption. Many cryptanalytic algorithms have time-memory trade-offs. However, accessing memory comes at a cost in time and energy. There is no universally accepted model for how random memory access and memory resources should be converted to computational cost

and resources. Often, computational cost and memory cost are reported separately, making it hard to directly compare computationally-heavy algorithms with memory-heavy algorithms or trade-offs between the two.

Bernstein and Chou address these issues with the CAT framework [BC24] by providing a concrete bit-operation cost model that covers both computation and memory in a unified model. They represent cryptanalytic algorithms as concrete Boolean circuits with a concrete implementation using basic bit-level primitives such as AND, XOR, and NOT gates. For example, a field addition over GF(256) is represented as a fixed XOR circuit on eight bits.

The model does not provide addressed memory as a unit-cost primitive. Intermediate results whose locations are known at circuit-generation time can be accessed directly, as if they were stored in named registers. This maps directly to static access patterns. However, runtime-dependent "random access" to such named registers must itself be implemented as a Boolean circuit. In CAT, this requires address arithmetic and a binary multiplexer tree for selecting the target register among the candidate registers. Hence, selecting one element of width w from n candidate values is modeled as a binary selection-tree circuit with cost $O(wn)$ bit operations. Thus, memory access has a concrete bit-operation cost and the cost of arithmetic and data access is expressed in the same cost model.

Alongside the circuit implementation, CAT uses fine-grained cost formulas to calculate concrete bit-operation counts for specific problem sizes. The CAT framework executes the circuits for small, tractable parameter sizes to verify the formulas on moderate input sizes to provide confidence in the formulas for cryptographically relevant instances.

This model provides a concrete interpretation of standard bit-security levels. While k-bit security is traditionally defined as requiring at least 2^k operations to break a system [Bar20], the underlying cost model is typically left implicit. By expressing cryptanalytic attacks as Boolean circuits and formalising their concrete cost in terms of elementary bit operations, the CAT framework provides a concrete metric that also includes random memory access. Concrete, precise, and compatible security levels can then be derived as the base-2 logarithm of the total bit-operation cost.

3 Circuit Design

Our XL circuit follows the algorithms described in Section 2 and uses Wiedemann linear algebra with Berlekamp–Massey sequence recovery. The implementation is written over an abstract finite-field interface, with concrete backends for GF(2), GF(31), and GF(256) built from the CAT bit class. For GF(31), we use schoolbook multiplication, while for GF(256) we use Bernstein's multiplication circuit[4], which minimizes the number of bit operations. Inversions are implemented using Fermat's little theorem with minimal addition chains[5]. We also provide dedicated classes for constant field elements: operations between constants are evaluated

[4] `https://binary.cr.yp.to/m.html`

[5] `https://wwwhomes.uni-bielefeld.de/achim/addition_chain.html`

at circuit-generation time, while operations between variables and constants use specialized circuits such as addition chains for constant multiplication.

The main consideration when implementing XL as circuit is how to formalize the required randomness. XL requires randomness only for the random dropping of rows when forming the square linear system matrix used as input to Wiedemann. We decided to hard-code the row-pruning pattern into the circuit. To keep the matrix-vector multiplication cost independent of the row-pruning pattern, we retain all rows corresponding to the original input polynomials and only prune extended rows. This means that during the linear-algebra part of Wiedemann, all coordinates and all values in the Macaulay matrix are known at circuit-generation time, and all access to the data in the Macaulay matrix B and the vectors v_k are to constant addresses that do not encounter random access cost in the circuit model.

Baseline variant. For our baseline implementation, we consider the polynomial system as the input to our XL circuit and the output (on success) is its solution that can be checked in polynomial time. If the computation fails, which is likely for small fields, we do not need to re-randomize the entire circuit. Instead, it is sufficient to randomly permute the polynomials in the input and to reuse the same circuit with the same row-pruning pattern.

Const variant. We also investigate a problem-specific variant where the circuit is specialized to a fixed polynomial system. This means that the Macaulay matrix is known at circuit-generation time and is treated as constant. The circuit therefore has no variable polynomial-system input; its output is the solution to the fixed, hard-coded system. This reduces the arithmetic cost in the Wiedemann phase, because all matrix-vector multiplications use multiplication by fixed field constants.

Const+bucket variant. Treating the data in the Macaulay matrix as constant also enables bucketed matrix-vector multiplication. The well-known idea is to group vector entries according to the corresponding matrix coefficient, first summing all entries with the same coefficient and then multiplying each bucket sum once by that coefficient. For variable coefficients this is not efficient in the CAT model: accumulating into coefficient-indexed buckets would require runtime-dependent access to the bucket table, whose selection cost exceeds the saving from avoiding multiplications. When the coefficients are fixed constants, however, the bucket assignment is known at circuit-generation time. The lookup is then static, and bucketing gives a substantial reduction in the matrix-evaluation cost.

Algorithmic optimization. We use a fixed-coordinate projection when forming the scalar sequence in Wiedemann's algorithm. In the notation of Algorithm 2, let $v_k = B^k b$ be the current Krylov vector. The scalar sequence is usually written as $s_k = u^\top v_k$. Instead of using a dense projection vector u, our implementation takes u to be a standard basis vector. Thus, s_k is one fixed coordinate of v_k. This avoids computing the full sum $u^\top v_k$ in each Wiedemann iteration, so the

projection does not add a separate cost term to the implementation model. This is analogous to the projection optimization used in block-Wiedemann implementations, e.g., [CCNY12, Sect. 3].

We also use an unpublished trick by Tung Chou[6]: Usually, when Wiedemann is used to solve a linear system, one is interested in recovering the full solution vector. In the context of XL, however, we only need the entries corresponding to the monomials of the original polynomial system. The remaining entries merely encode additional dependencies introduced by the Macaulay matrix. Therefore, Chou suggests computing only the required entries of the solution from the sequence provided to block Berlekamp–Massey. We adopt this idea to the non-block Wiedemann setting and evaluate the resulting polynomial only on the subvector corresponding to the monomials of the polynomial system.

4 Cost Analysis

In this section, we analyze the cost of our circuit XL model. The analysis is organized in two layers. The first layer gives formulas for the finite-field operation cost of the XL–Wiedemann computation, covering additions, subtractions, multiplications, and inversions. The second layer transforms these formulas into bit-operation costs using the concrete arithmetic circuits implemented for the fields used in our model. This separation is important because the Macaulay matrix and the Wiedemann algorithm determine the field-operation cost, while the representation of GF(2), GF(31), and GF(256) determines the bit-operation cost.

Before introducing bit-operation cost formulas, we first keep the arithmetic at the field level. This allows the XL and Wiedemann costs to be written independently of the different models used for GF(2), GF(31), or GF(256). Once the field-operation cost formulas are fixed, the next step evaluates these formulas using the bit-operation costs of the implemented field arithmetic.

4.1 Field-Operation Cost Formulas

At the field-operation level, one operation in the base field is counted as one unit. This keeps the combinatorial part of XL separate from the circuit realization of the field arithmetic. The formulas below therefore describe how many field additions, subtractions, multiplications, and inversions are used by our circuit XL model. We denote the corresponding totals by

$$A, \qquad S, \qquad M, \qquad I,$$

where the letters denote, in this order, the number of additions, subtractions, multiplications, and inversions over GF(q).

Let m denote the number of quadratic equations in the system with n variables, and d the operating degree used in the XL algorithm. The operating

[6] See: `https://www.polycephaly.org/projects/xl/index.shtml`

degree is chosen using the ordinary Yang–Chen estimate in the ordinary case. In the square-free GF(2) case, we instead use the corresponding square-free estimate. At this degree, the generated Macaulay matrix may have more rows than columns. We keep only enough rows to obtain the square system required by Wiedemann and discard the extra rows.

The cost formulas mainly depend on two values obtained from the pruned Macaulay matrix. The first one is the number of rows kept after pruning; we denote it by R for the ordinary cases (GF(31) and GF(256)), and by R_2 for the square-free case (GF(2)). The Macaulay matrix is very sparse with many structural zero entries. The only structurally populated entries are those corresponding to the original coefficients of the input system. We denote this number by Z for the ordinary case, and by Z_2 for the square-free case.

We first give the ordinary monomial-basis case, where the monomial basis is not reduced by square-free relations. For the chosen operating degree d, the full monomial basis has $\binom{n+d}{d}$ columns. In the non-homogeneous case, the constant column is moved to the right-hand side vector b. This is because the constant monomial is always 1, so it is not an unknown monomial value. More concretely, an equation with constant term is written as

$$\sum_{\alpha \neq 0} c_\alpha x^\alpha = -c_0,$$

so the constant coefficient is stored in b. Therefore, the left-hand Macaulay matrix contains only the nonconstant monomial columns and has $\binom{n+d}{d}-1$ columns. For the ordinary case, we write

$$R(n,d) = \binom{n+d}{d} - 1$$

for the final row count after pruning. The generated matrix has at least this many rows, so we prune the rows until $R(n,d)$ rows remain. We write

$$Z(n,m,d) = \frac{mn(n+3)}{2} + \frac{(R-m)(n+1)(n+2)}{2}$$

for the total number of stored matrix entries. We always retain the first m rows corresponding to the original input polynomials. Since the constant terms of these rows are moved to the right-hand side, pruning such rows would make the baseline cost depend on how many of these rows remain. This would introduce row-pruning-dependent cost variation and thus would make the closed formula an expected cost rather than an exact count. To keep the baseline formula exact, row pruning is restricted to the extended rows.

For the square-free case, the monomial count is obtained using the relation $x_i^2 = x_i$. Thus we write

$$B_d(n) = \sum_{k=0}^{d} \binom{n}{k}$$

for the number of square-free monomials of degree at most d, and set $B_d(n) = 0$ for $d < 0$. Then

$$R_2(n,d) = B_d(n) - 1,$$

$$T_2(n,m,d) = \Big(B_{d-2}(n) - 1 + n\big(B_{d-2}(n-1) - 1\big) + \frac{n(n-1)}{2}\big(B_{d-2}(n-2) - 1\big)\Big)\frac{R_2(n,d) - m}{B_{d-2}(n) - 1},$$

and

$$Z_2(n,m,d) = m\left(n + \frac{n(n-1)}{2}\right) + T_2.$$

Here R_2 is the number of rows after pruning. T_2 gives the structurally populated entries kept from the non-initial rows, and Z_2 is the total number of nonzero entries in the pruned Macaulay matrix. Since after the reduction $x_i^2 = x_i$, the extended rows do not have all the same number of nonzero entries and the row size changes with the multiplier degree, we use the averaged number of nonzero entries in the non-initial rows and then keep the same row proportion as in the pruning step.

In addition to considering different finite fields, we evaluate the cost under three arithmetic models: baseline, const, and const+bucket. In the baseline model, the entries of the Macaulay matrix are treated as ordinary field elements, so every scalar multiplication in the sparse linear-algebra stage is charged as a generic field multiplication. The const model uses the fact that the matrix coefficients are fixed after the Macaulay matrix has been constructed. Multiplication by such a fixed coefficient is then evaluated by a constant-coefficient multiplication formula rather than by a generic field multiplier. The const+bucket model goes one step further: during each sparse matrix evaluation, terms with the same coefficient are first grouped together, and the fixed coefficient is applied once to the accumulated bucket. Accordingly, the bucketed model changes only the matrix-evaluation cost in the Wiedemann algorithm; the costs of Berlekamp–Massey, normalization, and evaluation remain unchanged.

Our circuit XL model follows the same main structure in both the implementation and the cost formulas. Starting from the input quadratic system, the computation first generates the XL/Macaulay system at the chosen degree and forms the corresponding sparse matrix. After pruning, the Wiedemann step applies repeated sparse matrix evaluations to produce the scalar sequence. Berlekamp–Massey is then applied to this sequence, and the recovered relation is finally normalized and evaluated to obtain the solution candidate. While deriving the cost formulas, we follow this same order and write the contribution of each stage separately.

Other than differences in how R and R_2 as well as Z and Z_2 are computed, the cost formulas for the square-free case and the ordinary case are the same. Therefore, we write R_q and Z_q in the following, to be specialized for the concrete field.

The computations for Berlekamp–Massey as well as the normalization and the final evaluation of $\mu(x)$ are the same for all variants and have the following cost:

	A	S	M	I
Berlekamp–Massey	$\frac{3\,(R_q+1)R_q}{2}$	$\frac{(3\,R_q+5)R_q}{2}$	$\frac{(9\,R_q+13)R_q}{2}$	0
normalization	0	0	R_q+1	1
evaluation	$R_q n$	0	$R_q n$	0

The cost for the linear algebra of Wiedemann changes for the different variants. We now have a new cost term M^{fix} for multiplication by a constant and the baseline multiplication count now becomes constant. For const+bucket, the number of additions and multiplications changes more significantly. In each matrix evaluation, entries with the same nonzero coefficient are first accumulated into a bucket. After the bucket sums are formed, each nonzero coefficient is applied once to its bucket. Thus, there are $q-2$ nontrivial coefficient multiplications per matrix-vector product. The operation counts are:

	A	M	M_{fixed}
baseline	$(2\,R_q-1)Z_q$	$(2\,R_q-1)Z_q$	0
const	$(2\,R_q-1)Z_q$	0	$(2\,R_q-1)Z_q$
const+bucket	$\left(R_q(q-2)+\frac{Z_q(q-1)}{q}\right)(2\,R_q-1)$	0	$(2\,R_q-1)R_q(q-2)$

Adding up all the operation counts for Berlekamp–Massey, normalization, and evaluation as well as the Wiedemann part for the baseline variant, we get:

$$
\begin{aligned}
A^{\text{base}} &= \frac{3}{2}\,(R_q+1)R_q + (2\,R_q-1)Z_q + R_q n \\
S^{\text{base}} &= \frac{1}{2}\,(3\,R_q+5)R_q \\
M^{\text{base}} &= (3\,R_q+5)R_q + \frac{3}{2}\,(R_q+1)R_q + (2\,R_q-1)Z_q + R_q n + R_q + 1 \\
I^{\text{base}} &= 1
\end{aligned}
$$

For the const variant, we get:

$$
\begin{aligned}
A^{\text{const}} &= \frac{3}{2}\,(R_q+1)R_q + (2\,R_q-1)Z_q + R_q n \\
S^{\text{const}} &= \frac{1}{2}\,(3\,R_q+5)R_q \\
M^{\text{const}} &= (3\,R_q+5)R_q + \frac{3}{2}\,(R_q+1)R_q + R_q n + R_q + 1 \\
M^{\text{const}}_{\text{fixed}} &= (2\,R_q-1)Z_q \\
I^{\text{const}} &= 1
\end{aligned}
$$

Finally, the const+bucket variant adds up to:

$$\begin{aligned}
A^{\text{buck}} &= \left(R_q(q-2) + \frac{Z_q(q-1)}{q}\right)(2\,R_q - 1) + \frac{3}{2}\,(R_q+1)R_q + R_q n \\
S^{\text{buck}} &= \frac{1}{2}\,(3\,R_q + 5)R_q \\
M^{\text{buck}} &= (3\,R_q + 5)R_q + \frac{3}{2}\,(R_q+1)R_q + R_q n + R_q + 1 \\
M^{\text{buck}}_{\text{fixed}} &= (2\,R_q - 1)R_q(q-2) \\
I^{\text{buck}} &= 1
\end{aligned}$$

4.2 Bit-Operation Cost Formulas

The bit-operation cost formulas take the field-operation formulas from the previous subsection and assign a bit-operation cost to each field operation. They do not change the algebraic structure of the circuit XL model; the same totals A, S, M, and I are used. What changes is the unit in which the cost is measured. For example, addition in GF(2) is one XOR operation, while addition in GF(31) and GF(256) has a field-specific bit-operation cost, with an explicit reduction step. Thus, the bit-operation formulas keep the same field-operation structure and weight each arithmetic operation by the corresponding bit-operation cost for the field. This separation matters because the same sparse linear-algebra schedule can lead to quite different costs over different fields. The support of the Macaulay matrix determines how many field operations are requested, whereas the field representation determines how expensive each requested operation is in bits. The formulas below follow exactly this convention: first the bit-operation cost of the primitive field arithmetic is fixed, then the field-operation totals from the previous subsection are substituted into the corresponding bit model.

The extra cost comes from the bit operations inside Berlekamp–Massey, which are not included in the field-operation cost formulas. These include comparisons, bit-vector updates, and conditional selections that are needed by the Berlekamp–Massey step but are not themselves field operations. These operations are performed on bit-vectors whose length depends on the sequence length. Since the Wiedemann sequence has length proportional to $2R$ in the ordinary case and $2R_2$ in the square-free GF(2) case, we define

$$\ell = \lceil \log_2(2R) \rceil\,, \qquad \ell_2 = \lceil \log_2(2R_2) \rceil\,.$$

Here ℓ and ℓ_2 give the number of bits needed to represent the indices used inside Berlekamp–Massey. The additional bit-operation cost of this non-field part is

$$\begin{aligned}
E_2 &= \frac{1}{2}\,(9\,R_2 + 56\,\ell_2 - 25)R_2, \\
E_{31} &= \frac{1}{2}\,(45\,R + 56\,\ell + 99)R, \quad \text{and} \\
E_{256} &= 4\,(9\,R + 7\,\ell + 24)R.
\end{aligned}$$

Field	q	add a_q	sub s_q	mul m_q	inv i_q	mul const μ_q	bucket mul const $\mu_q q/(q-2)$
GF(2)	2	1	1	1	0	0	–
GF(31)	31	41	46	207	1449	153.74	164.34
GF(256)	256	8	8	132	752	38.75	39.06

Table 1: Bit-operation costs of the field operations used in our model. Here a_q, s_q, m_q, i_q denote the costs of field addition, subtraction, multiplication, and inversion in $\mathbb{F}_q$, respectively. The column μ_q is the average cost of multiplication by a fixed field constant in $\mathbb{F}_q$. The bucket-mul-const column gives the conditional cost $\mu_q q/(q-2)$ for constants $2, \ldots, q-1$, used only in the const+bucket model for $q > 2$; for $q = 2$ no such multiplications occur.

The primitive bit-operation costs are listed in Table 1.

For both GF(31) and GF(256), the inversion used in the normalization step is kept out of the multiplication costs and charged as one primitive bit-operation inversion. In GF(31) the inversion is implemented by an addition chain with 7 field multiplications, so its bit-operation cost is 1449. In GF(256) the inversion is implemented by a chain with 7 squarings and 4 field multiplications, giving bit-operation cost 752.

The auxiliary quantity μ_q in Table 1 is used only for multiplications by fixed coefficients. For the const variants, only the matrix-coefficient multiplications in the Wiedemann matrix evaluations use μ_q. The other field multiplications, such as those in Berlekamp–Massey and normalization, are still charged with the normal mul factor.

In the const model, the fixed constants arise directly from the Macaulay matrix and are therefore weighted by μ_q. In the const+bucket model, however, the fixed multiplications in the bucket combination step are only by the nontrivial bucket coefficients $2, \ldots, q-1$. Hence, for $q > 2$, these multiplications are weighted by the conditional average

$$\mu_{q,>1} = \frac{q}{q-2}\mu_q.$$

For $q = 2$, no such bucket multiplications occur. Thus, whenever $M_{\text{fixed}}^{\text{buck}}$ appears in a bit-cost formula, its contribution is

$$M_{\text{fixed}}^{\text{buck}} \cdot \mu_q \frac{q}{q-2}.$$

Overall, we obtain the following bit-operation costs by multiplying the operation counts $A, S, M, M_{\text{fixed}}^{\text{const}}, M_{\text{fixed}}^{\text{buck}}, I$ by the corresponding bit-operation costs from Table 1 and adding the extra Berlekamp–Massey term E_q.

GF(2), baseline model.

$$B_2^{\text{base}} = 12\,R_2^2 + 4\,R_2 Z_2 + 28\,R_2 \ell_2 + 2\,R_2 n - R_2 - 2\,Z_2 + 1$$

GF(2), const model.

$$B_2^{\text{const}} = 12\,R_2^2 + 2\,R_2 Z_2 + 28\,R_2\ell_2 + 2\,R_2 n - R_2 - Z_2 + 1$$

GF(2), const+bucket model.

$$B_2^{\text{buck}} = 12\,R_2^2 + R_2 Z_2 + 28\,R_2\ell_2 + 2\,R_2 n - R_2 - \frac{1}{2}\,Z_2 + 1$$

GF(31), baseline model.

$$\begin{aligned} C_{31}^{\text{base}} = {} & \frac{2169}{2}\,R_{31}^2 + 496\,R_{31}Z_{31} + 28\,R_{31}\ell + 248\,R_{31}n \\ & + \frac{3557}{2}\,R_{31} - 248\,Z_{31} + 1656 \end{aligned}$$

GF(31), const model.

$$\begin{aligned} C_{31}^{\text{const}} = {} & 2\,R_{31}Z_{31}\mu_{31} + \frac{2169}{2}\,R_{31}^2 + 82\,R_{31}Z_{31} + 28\,R_{31}\ell - Z_{31}\mu_{31} \\ & + 248\,R_{31}n + \frac{3557}{2}\,R_{31} - 41\,Z_{31} + 1656 \end{aligned}$$

GF(31), const+bucket model.

$$\begin{aligned} C_{31}^{\text{buck}} = {} & 58\,R_{31}^2\mu_{31}\frac{31}{29} + \frac{6925}{2}\,R_{31}^2 + \frac{2460}{31}\,R_{31}Z_{31} + 28\,R_{31}\ell - 29\,R_{31}\mu_{31}\frac{31}{29} \\ & + 248\,R_{31}n + \frac{1179}{2}\,R_{31} - \frac{1230}{31}\,Z_{31} + 1656 \end{aligned}$$

GF(256), baseline model.

$$\begin{aligned} C_{256}^{\text{base}} = {} & 654\,R_{256}^2 + 280\,R_{256}Z_{256} + 28\,R_{256}\ell + 140\,R_{256}n \\ & + 1118\,R_{256} - 140\,Z_{256} + 884 \end{aligned}$$

GF(256), const model.

$$\begin{aligned} C_{256}^{\text{const}} = {} & 2\,R_{256}Z_{256}\mu_{256} + 654\,R_{256}^2 + 16\,R_{256}Z_{256} + 28\,R_{256}\ell - Z_{256}\mu_{256} \\ & + 140\,R_{256}n + 1118\,R_{256} - 8\,Z_{256} + 884 \end{aligned}$$

GF(256), const+bucket model.

$$\begin{aligned} C_{256}^{\text{buck}} = {} & 508\,R_{256}^2\mu_{256}\frac{256}{254} + 4718\,R_{256}^2 + \frac{255}{16}\,R_{256}Z_{256} + 28\,R_{256}\ell \\ & - 254\,R_{256}\mu_{256}\frac{256}{254} + 140\,R_{256}n - 914\,R_{256} - \frac{255}{32}\,Z_{256} + 884 \end{aligned}$$

5 Evaluation

In this section, we evaluate our concrete XL cost model. We first validate the derived formulas against circuit executions on small parameter sizes. We then study the asymptotic behavior of the concrete costs and their convergence to the expected leading constant factors. Finally, we apply the model to parameter sets from the Fukuoka MQ Challenge and selected multivariate candidates in the NIST additional-signature process.

5.1 Formula Accuracy

To validate the accuracy of our cost formulas, we evaluated the three XL variants baseline (base), const (c), and const+bucket (c-b) over the fields GF(2), GF(31), and GF(256). We considered the parameter regime $m = 2n$ and selected six representative parameter sets covering both $D = 3$ and $D = 4$. For GF(31) and GF(256), we used $n \in \{4, \ldots, 9\}$, while for GF(2) we used $n \in \{10, \ldots, 15\}$. For each configuration, we performed 100 independent runs with random input systems.

Figure 1 shows that the predicted and measured costs agree closely across all tested configurations. For the baseline variants over GF(31) and GF(256), the relative prediction error is zero, confirming that the formulas account accurately for the concrete bit-operation costs of our circuit implementation. For the const variants, small data-dependent fluctuations appear, because the concrete cost depends on the distribution of field coefficients in the randomly generated input systems, while the formulas predict the expected cost averaged over the field elements. These effects are substantially smaller for the const+bucket variants, since data-dependent operations here only appear for multiplications by zero and one. Hence, the variation decreases with increasing field size and is smallest for GF(256).

The GF(2) experiments show an additional systematic offset caused by the Boolean reduction $x_i^2 = x_i$. After reduction, rows generated by multiplying with different monomials contain different numbers of surviving entries. Our formulas model this effect using averaged row counts, while the concrete experiments use fixed row selections determined by the circuit generated for each problem size independently of the input data. This produces a small but consistent relative error for the baseline and const variants. For the const+bucket variant, the effect observed for GF(31) and GF(256) is more pronounced, because the data dependency of operations involving the constants zero and one for GF(2) affects all field operations, whereas in larger fields only a small fraction of coefficients are affected.

Table 2 summarizes the aggregate prediction error over all parameter sets. Although some isolated runs show comparatively large deviations, the mean relative error remains centered near zero with very low standard deviation. This confirms that the remaining discrepancies are caused by data-dependent execution effects rather than inaccuracies in the cost formulas themselves.

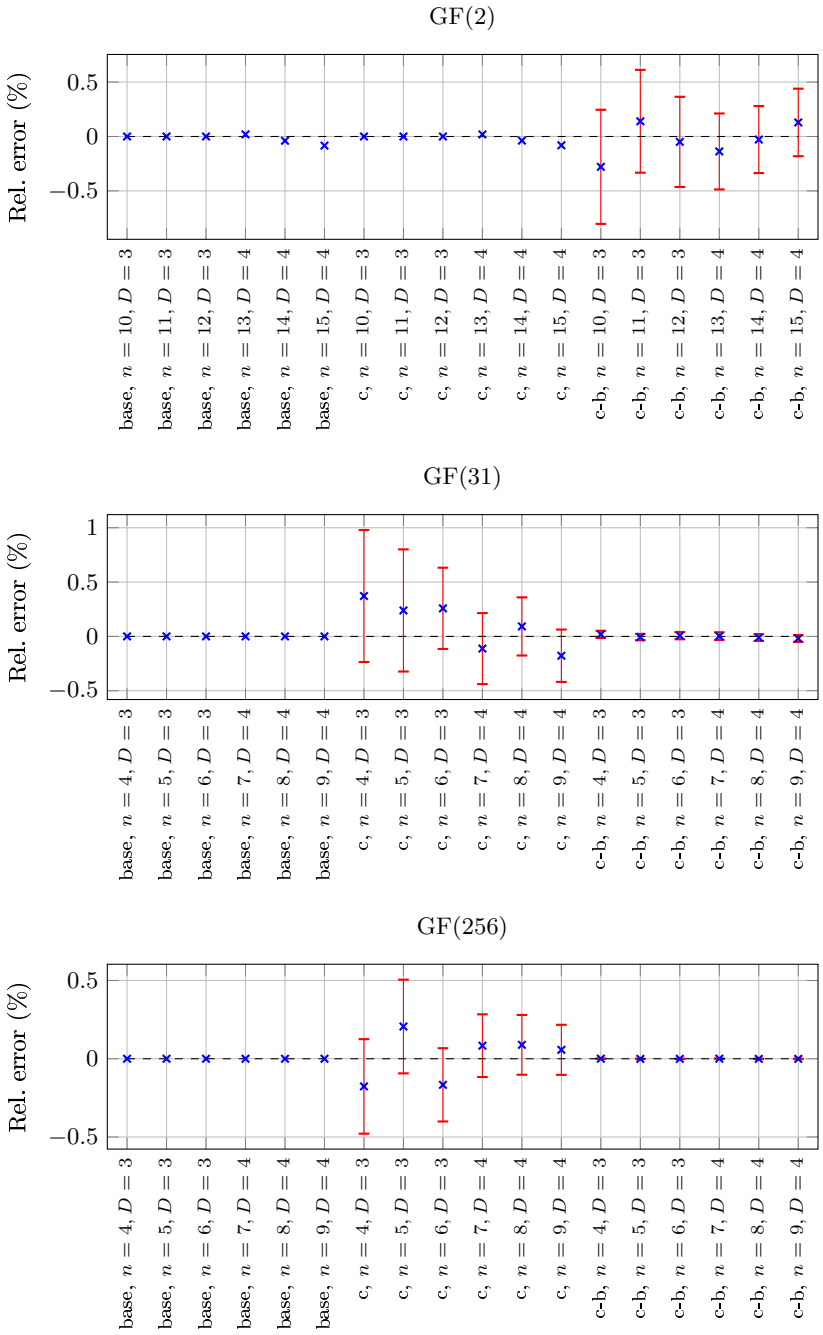

Fig. 1: Relative prediction error separated by field size. Each point is the mean over 100 independent random seeds; error bars show 95% confidence intervals. Data-dependent effects are expected to be strongest over GF(2) and weakest over GF(256).

Field	Variant	Cases: n	Runs/case	Mean rel. err.	Range
GF(2)	baseline	10–15	100	−0.017 (±0.038%)	−0.083% — 0.020%
GF(2)	const	10–15	100	−0.017 (±0.036%)	−0.081% — 0.019%
GF(2)	const+bucket	10–15	100	−0.037 (±0.160%)	−5.726% — 7.348%
GF(31)	baseline	4–9	100	0.000 (±0.000%)	0.000%
GF(31)	const	4–9	100	0.112 (±0.219%)	−6.589% — 7.530%
GF(31)	const+bucket	4–9	100	−0.001 (±0.013%)	−0.436% — 0.523%
GF(256)	baseline	4–9	100	0.000 (±0.000%)	0.000%
GF(256)	const	4–9	100	0.015 (±0.154%)	−3.261% — 4.705%
GF(256)	const+bucket	4–9	100	−0.000 (±0.001%)	−0.014% — 0.027%

Table 2: Prediction error grouped by field size and variant with mean relative error (± standard deviation) and range (min-max) across cases. Each case is averaged over 100 random seeds. Errors are reported as (predicted − measured)/measured.

5.2 Asymptotic Cost Behavior

Figure 2 shows the bit-operation complexity predicted by our concrete cost formulas for solving quadratic systems with XL over the fields GF(2), GF(31), and GF(256) for the three implementation variants baseline, const, and const+bucket, using the parameter regime $m = 2n$.

For reference, we also plot the leading-order complexity estimate given by:

$$\begin{cases} 2R_2(n,d)Z_2(n,m,d) & \text{for GF(2)} \\ 2R(n,d)Z(n,m,d) & \text{for GF(31) and GF(256)} \end{cases}$$

where R is the number of matrix rows, giving $2R$ Wiedemann iterations. Z denotes the number of entries structurally induced by the extended input system, i.e., entries arising from monomial multiples of the original input polynomials. The reference curve is not intended as a big-O bound, but as a leading-order cost estimate that includes the iteration-count factor used in the standard XL cost heuristic.

Across all fields, the curves exhibit essentially identical asymptotic scaling behavior. The staircase structure is caused by discrete changes in the effective Macaulay degree that cause abrupt increases in the dimensions of the generated Macaulay matrices. The enlarged insets provide a more detailed view on the relative cost between different variants.

The three implementation variants differ primarily by concrete constant factors. In particular, the optimized variants reduce the effective bit-operation cost while preserving the same asymptotic scaling behavior. This behavior is consistent with the fact that the dominant contribution to the XL complexity arises from the Wiedemann linear-algebra phase, whose cost is determined asymptoti-

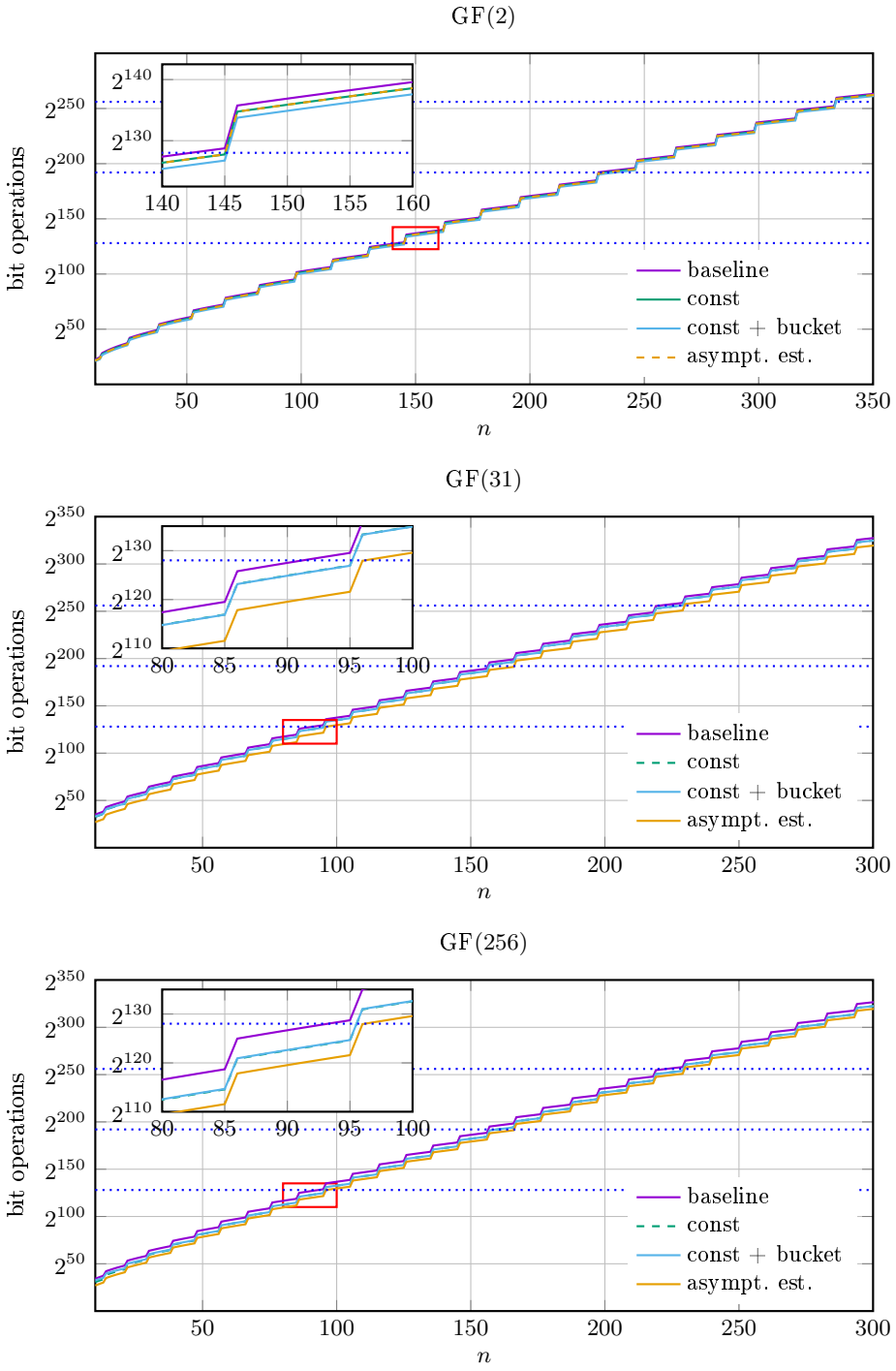

Fig. 2: Cost for GF(2), GF(31), and GF(256) with $m = 2n$.

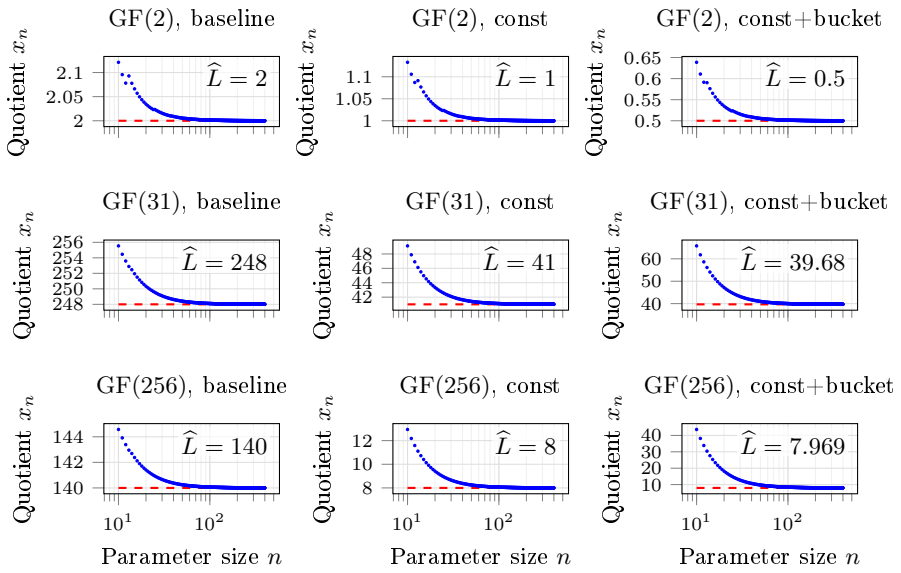

Fig. 3: Quotients x_n as functions of the parameter size n for the three implementation variants over GF(2), GF(31), and GF(256). Dashed lines indicate the asymptotic constant factors $\widehat{L}$ obtained from least-squares fits of the form $x_n = \widehat{L} + Cn^{-2}$.

cally by the number of matrix-vector operations together with the concrete costs of field additions and multiplications.

For GF(2), the const+bucket implementation occasionally yields a concrete cost marginally below the dominant-term estimate. This reflects concrete implementation effects and lower-order terms rather than a difference in the dominant asymptotic scaling.

5.3 Convergence Analysis

Figure 3 shows the quotient between the predicted implementation cost and the corresponding leading-order complexity estimate in order to identify the concrete constant factors relating them. Our expectation is that the concrete cost differs from the asymptotic estimate only by a field- and implementation-dependent multiplicative constant together with lower-order terms such as the cost for Berlekamp–Massey. The results confirm this expectation: in every configuration, the quotient converges toward a stable limiting constant $\widehat{L}$ as the parameter size n increases.

For the baseline implementation, the observed limiting constants coincide exactly with the sum of the bit-operation costs of field addition and multiplication listed in Table 1. More precisely,

$$\widehat{L}_{\text{base}} = a_q + m_q,$$

yielding the limiting values $1 + 1 = 2$, $41 + 207 = 248$, and $8 + 132 = 140$ for GF(2), GF(31), and GF(256), respectively. This confirms the expectation that the asymptotic XL cost is dominated by the Wiedemann linear-algebra phase and that the leading concrete factor is determined by the underlying field arithmetic.

For the const variant, we get the limiting factor from the column μ_q in Table 1, which lists the average bit-cost of multiplication by a fixed field element averaged uniformly over all field elements, as:

$$\widehat{L}_{\text{const}} = a_q + \mu_q.$$

For the const+bucket variant, multiplication is mostly replaced by bucketed additions. The remaining fixed multiplications by bucket coefficients contribute only lower-order terms to the quotient considered here. Therefore, the limiting factor is determined by the average cost of adding a nonzero bucket contribution:

$$\widehat{L}_{\text{const+bucket}} = \frac{q-1}{q}\, a_q.$$

The exact agreement between the observed limiting constants and the predicted bit-operation costs supports the expectation that the asymptotic XL behavior is fully dominated by the Wiedemann phase.

In all configurations, the data is well approximated by a straight line in log–log representation, indicating asymptotic power-law decay of the form

$$|x_n - \widehat{L}| \sim C n^{-\alpha}.$$

Least-squares fits consistently yield

$$\alpha \approx 2,$$

indicating that the lower-order discrepancy between the concrete costs and the leading-order estimate decays quadratically in n.

5.4 Fukuoka MQ Challenge Cost Estimates

For concrete security estimates of the Fukuoka MQ Challenges, we apply the hybrid-XL strategy from Section 2.1, which guesses variables before applying XL. If g variables are guessed, the reduced system has $n' = n - g$ variables and still m equations. For underdetermined systems, $m < n$, the first $n - m$ specializations remove the excess variables and bring the system to $n' = m$. We do not count these specializations in the main exhaustive-search exponent of the hybrid cost model. Additional guesses are charged by a multiplicative factor q^k, where

$$k = g - \max\{0, n - m\}.$$

In our Wiedemann-XL model, however, we only evaluate reduced systems satisfying $m \geq n' + 2$. Thus, for initially underdetermined systems, $m < n$, the

Type	n	D	k	base	c+b	Type	m	D	k	base	c+b
	largest broken challenges						*largest broken challenges*				
Type I GF(2) $m = 2n$	80	6	17	82.14	80.15	**Type IV** GF(2) $n \approx 1.5m$	75	3	54	84.96	83.02
	83	6	19	84.47	82.47		76	3	55	85.96	84.02
	largest provided challenges						*largest provided challenges*				
	99	7	19	95.89	93.90		99	4	65	107.45	105.47
	100	7	20	96.89	94.90		100	4	66	108.45	106.47
	largest broken challenges						*largest broken challenges*				
Type II GF(256) $m = 2n$	36	7	0	67.47	65.01	**Type V** GF(256) $n \approx 1.5m$	19	10	2	77.58	76.75
	37	7	0	68.05	65.53		20	11	2	81.81	80.84
	largest provided challenges						*largest provided challenges*				
	53	9	0	87.15	84.04		34	15	3	118.97	116.80
	54	9	0	87.64	84.51		35	18	2	122.68	120.39
	largest broken challenges						*largest broken challenges*				
Type III GF(31) $m = 2n$	37	7	0	68.88	66.51	**Type VI** GF(31) $n \approx 1.5m$	23	10	3	81.36	79.48
	38	7	0	69.44	67.06		24	8	5	83.62	81.79
	largest provided challenges						*largest provided challenges*				
	52	9	0	87.47	84.98		34	12	5	108.35	106.14
	53	9	0	87.97	85.48		35	14	4	112.38	110.12

Table 3: Concrete XL cost estimates for the largest listed Fukuoka MQ Challenge parameter sets for the baseline and the const+bucket (c+b) variants. In all cases, D and k are the same for both variants, so we report them only once.

starting point in our search is $g = n - m + 2$, corresponding to $k = 2$. We then minimize the total cost

$$C_{\text{hybrid}} = q^k C_q(n', m)$$

over all applicable guesses. For already overdetermined systems, $m \geq n$, there are no free specializations, so $k = g$.

Table 3 reports concrete XL cost estimates for some parameter sets of the Fukuoka MQ Challenge. We report the largest two broken challenges and the two largest provided parameter sets (available at the time of writing). For each challenge type, we provide cost predictions for the baseline variant and for the const+bucket variant; the intermediate const variant is omitted for brevity. All costs are given as base-two logarithms of bit operations.

Several of the challenges have been solved using approaches other than XL: Types I and IV with hybrid XL variants, Types II, V, and VI with F4/F5 solvers, and only Type III with an XL approach using block Wiedemann. Given our cost model, the broken Type II and III challenges correspond to 2^{67} to 2^{69} bit operations. The broken Type I, IV, V, and VI challenges lie in a substantially higher complexity range of approximately 2^{77} to 2^{86} bit operations in our cost model.

Our estimates provide a common hybrid-XL baseline for comparing the challenge families, but they are not intended to predict the cost of the best known specialized attacks.

The overdetermined Fukuoka MQ Challenge families reach approximately 2^{84} to 2^{97} bit operations under our cost model, while the underdetermined challenges reach approximately 2^{105} to 2^{123} bit operations.

5.5 Multivariate Signature Candidates

A natural application of our cost model is the analysis of multivariate candidates in the NIST additional-signature process. The Round 2 candidates in this category are MAYO, QR-UOV, SNOVA, UOV, and MQOM. Although their internal structures differ, the corresponding systems exposed to direct algebraic attacks consist of quadratic equations over finite fields. For the fields currently implemented in our model, the most direct comparison targets are MQOM over GF(256), UOV over GF(256), and QR-UOV over GF(31). MAYO and SNOVA are also relevant multivariate candidates, but they require a GF(16) backend and are therefore left for future work.

The MQ-based signature schemes considered here satisfy $m \leq n$, so we apply the hybrid-XL strategy from Section 2.1. For each parameter set, we specialize variables until $m \geq n' + 2$ and minimize the resulting total cost over additional guesses. The table reports the minimizing value of k, where k counts guesses beyond the square-system threshold $n' = m$, giving

$$C_{\text{hybrid}} = q^k C_q.$$

The security analysis of MQOM is based on the unstructured MQ problem [BFR24], and the MQOM submission document Version 2.1 [FR25] gives parameters for the hybrid-XL-Wiedemann algorithm, denoted WXL, over GF(256). The MQOM estimates use block Wiedemann, whereas our implementation uses the scalar Wiedemann setting. Table 4 shows the resulting costs, including the value for k that minimizes the total cost. Our circuit model as well as the MQOM security analysis explicitly account for the bit-operation costs of individual field operations. As a result, our base model yields almost the same cost estimate as the MQOM analysis and overall our models are consistent with the claimed security levels of MQOM.

Table 4 additionally includes parameter sets for UOV and QR-UOV. UOV is the classical oil-and-vinegar construction [KPG99], while QR-UOV is a quotient-ring variant of UOV [FIKT21]. The security estimates by the UOV team for Wiedemann-XL include an approximation of the field-arithmetic cost, namely $\lceil \log_2(q) \rceil^2$ bit operations for multiplication and $\lceil \log_2(q) \rceil$ bit operations for addition. Therefore, the resulting estimates are very close to those obtained from our concrete circuit model. The QR-UOV estimates do not include bit-operation costs for field operations. Consequently, the published QR-UOV estimates are lower than ours by the field-arithmetic cost factors discussed in Section 5.3.

Although MQOM, UOV, and QR-UOV all analyze Wiedemann-XL attacks, they employ different concrete cost models. MQOM and UOV use effectively the

Scheme	q	n	m	WXL	base	const	c+b	D	k
MQOM2-L1-gf256	256	48	48	157	158.0	156.5	155.2	22	3
MQOM2-L3-gf256	256	72	72	217.5	218.6	217.0	215.2	28	5
MQOM2-L5-gf256	256	96	96	280.4	281.4	279.8	277.8	33	8
uov-Ip	256	112	44	145	146.9	145.3	144.2	20	3
uov-III	256	184	72	218	218.6	217.0	215.2	28	5
uov-V	256	244	96	278	281.4	279.8	277.8	33	8
QR-UOV I (31, 165, 60, 3)	31	75	60	163	171.3	171.0	168.8	20	7
QR-UOV III (31, 246, 87, 3)	31	111	87	226	234.7	234.4	232.2	28	9
QR-UOV V (31, 324, 114, 3)	31	156	114	286	294.4	294.0	291.8	31	15

Table 4: Comparison of our concrete XL cost estimates with the published Wiedemann-XL (WXL) attack estimates for MQOM, UOV, and QR-UOV parameter sets. The column "WXL" lists the bit complexities reported in the corresponding scheme specifications. For our variants base, const, and const+bucket (c+b), D and k are the same, so we report them only once.

same (n, m) values for the L3/III and L5/V parameter sets (after guessing for UOV), but they derive different complexity estimates. In contrast, our formulas apply a uniform concrete bit-operation cost model across all three schemes, thereby enabling directly comparable attack estimates. From this perspective, the MQOM2-L1-gf256 as well as the QR-UOV parameter sets appear slightly conservative with respect to Wiedemann-XL attacks.

6 Conclusion

This work extends the CryptAttackTester methodology to XL-based algebraic cost analysis by giving concrete bit-operation formulas for XL with Wiedemann linear algebra and Berlekamp–Massey sequence recovery. The formulas separate the combinatorial structure of the Macaulay matrices from the bit-level cost of the underlying field arithmetic, which makes it possible to instantiate the same attack model over different fields and implementation variants. The validation experiments show that these formulas accurately predict the circuit costs, while the convergence analysis explains the observed constants in terms of the field-operation costs.

The comparison between NIST PQC signature candidates illustrates the usefulness of a unified cost model. Attack estimates that are all nominally based on Wiedemann-XL can differ because they account for field arithmetic and algorithmic optimizations differently. Expressing these costs in a common bit-operation model makes the resulting security estimates directly comparable and helps identify where parameter choices include additional margins with respect to XL-based attacks.

Future work should extend the finite-field backend to GF(16), which would enable direct analysis of further multivariate candidates such as MAYO and SNOVA. A broader comparison of algebraic attacks would also require implementing fast exhaustive search and crossbred attacks for GF(2), as well as Gröbner-basis algorithms such as F4 and F5 across the supported fields. This would move toward a unified bit-operation framework for evaluating algebraic cryptanalysis of multivariate schemes.

References

Bar20. Elaine B. Barker. Recommendation for key management: Part 1 – general. Special Publication (NIST SP) 800-57 Part 1 Revision 5, National Institute of Standards and Technology, May 2020.

BC24. Daniel J. Bernstein and Tung Chou. CryptAttackTester: high-assurance attack analysis. In *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part VI*, volume 14925 of *Lecture Notes in Computer Science*, pages 141–182. Springer, 2024.

BCC+10. Charles Bouillaguet, Hsieh-Chung Chen, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, Adi Shamir, and Bo-Yin Yang. Fast exhaustive search for polynomial systems in F_2. In *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2010.

Ber68. Elwyn R. Berlekamp. *Algebraic coding theory.* McGraw-Hill series in systems science. McGraw-Hill, 1968.

BFR24. Ryad Benadjila, Thibauld Feneuil, and Matthieu Rivain. MQ on my mind: Post-quantum signatures from the non-structured multivariate quadratic problem. In *9th IEEE European Symposium on Security and Privacy, EuroS&P 2024, Vienna, Austria, July 8-12, 2024*, pages 468–485. IEEE, 2024.

CCNY12. Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, and Bo-Yin Yang. Solving quadratic equations with XL on parallel architectures. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 356–373. Springer, Berlin, Heidelberg, September 2012.

CKPS00. Nicolas T. Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Springer, 2000.

Cop94. Don Coppersmith. Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm. *Math. Comput.*, 62(205):333–350, January 1994.

CS99. Hsie-Chia Chang and C. Bernard Shung. New serial architecture for the Berlekamp–Massey algorithm. *IEEE Trans. Commun.*, 47(4):481–483, 1999.

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

Fau02. Jean Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, page 75–83. Association for Computing Machinery, 2002.

FIKT21. Hiroki Furue, Yasuhiko Ikematsu, Yutaro Kiyomura, and Tsuyoshi Takagi. A new variant of unbalanced oil and vinegar using quotient ring: QR-UOV. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part IV*, Lecture Notes in Computer Science, pages 187–217. Springer, 2021.

FR25. Thibauld Feneuil and Matthieu Rivain. MQOM — MQ on my Mind – algorithm specifications and supporting documentation (version 2.1). Technical report, 2025. available at `https://mqom.org/docs/mqom-v2.1.pdf`.

JV17. Antoine Joux and Vanessa Vitse. A crossbred algorithm for solving Boolean polynomial systems. In Jerzy Kaczorowski, Josef Pieprzyk, and Jacek Pomykala, editors, *Number-Theoretic Methods in Cryptology - First International Conference, NuTMiC 2017*, Lecture Notes in Computer Science, pages 3–21. Springer, 2017.

KPG99. Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced oil and vinegar signature schemes. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, Lecture Notes in Computer Science, pages 206–222. Springer, 1999.

Mas69. James L. Massey. Shift-register synthesis and BCH decoding. *IEEE Trans. Inf. Theory*, 15(1):122–127, 1969.

Tho02. Emmanuel Thomé. Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm. *J. Symb. Comput.*, 33(5):757–775, 2002.

Wie86. Douglas H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory*, 32(1):54–62, 1986.

YC04a. Bo-Yin Yang and Jiun-Ming Chen. All in the XL family: Theory and practice. In *Information Security and Cryptology – ICISC 2004*, volume 3506 of *Lecture Notes in Computer Science*, pages 67–86. Springer, 2004.

YC04b. Bo-Yin Yang and Jiun-Ming Chen. Theoretical analysis of XL over small fields. In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *ACISP 04*, volume 3108 of *LNCS*, pages 277–288. Springer, Berlin, Heidelberg, July 2004.