Reducing the Number of Qubits in Quantum Information Set Decoding

Clémence Chevignard, Pierre-Alain Fouque, and André Schrottenloher

Univ Rennes, Inria, CNRS, IRISA, Rennes, France
firstname.lastname@inria.fr

Abstract. This paper presents an optimization of the memory cost of the quantum *Information Set Decoding* (ISD) algorithm proposed by Bernstein (PQCrypto 2010), obtained by combining Prange's ISD with Grover's quantum search.
When the code has constant rate and length n, this algorithm essentially performs a quantum search which, at each iteration, solves a linear system of dimension $\mathcal{O}(n)$. The typical code lengths used in post-quantum public-key cryptosystems range from 10^3 to 10^5. Gaussian elimination, which was used in previous works, needs $\mathcal{O}(n^2)$ space to represent the matrix, resulting in millions or billions of (logical) qubits for these schemes.
In this paper, we propose instead to use the algorithm for sparse matrix inversion of Wiedemann (IEEE Trans. inf. theory 1986). The interest of Wiedemann's method is that one relies only on the implementation of a matrix-vector product, where the matrix can be represented in an implicit way. This is the case here.
We give two main trade-offs, which we have fully implemented, tested on small instances, and benchmarked for larger instances. The first one is a quantum circuit using $\mathcal{O}(n)$ qubits, $\mathcal{O}(n^3)$ Toffoli gates like Gaussian elimination, and depth $\mathcal{O}(n^2 \log n)$. The second one is a quantum circuit using $\mathcal{O}(n \log^2 n)$ qubits, $\mathcal{O}(n^3)$ gates in total but only $\mathcal{O}(n^2 \log^2 n)$ Toffoli gates, which relies on a different representation of the search space.
As an example, for the smallest Classic McEliece parameters we estimate that the Quantum Prange's algorithm can run with 18098 qubits, while previous works would have required at least half a million qubits.

Keywords: Prange's Algorithm, Quantum Search, Information Set Decoding, Quantum Cryptanalysis.

1 Introduction

Since the start of the NIST's post-quantum cryptography standardization process [43], several cryptosystems based on error correcting codes came to light.

The remaining key-encapsulation candidates Classic McEliece [13], HQC [1] and BIKE [3] are currently in the fourth round [45]. Additionally, many more code-based cryptosystems have been proposed at the NIST's call for additional post-quantum signature schemes [44].

Cryptographic algorithms based on codes usually rely on the hardness of the Syndrome Decoding Problem (SDP). This problem essentially consists in finding a solution to an undetermined linear system, which is constrained under a given metric. In the cases considered on this paper, the Hamming metric is used, which counts the number of non-zero coordinates.

The most efficient algorithms to solve the SDP are the family of Information Set Decoding algorithms (ISD), starting with Prange's algorithm [48], which have been gradually improved over time with list-merging subroutines [53,38,40,6], nearest-neighbor techniques [41,16] and more recently sieving techniques [32,25]. However, all these optimizations essentially improve the time complexity at the detriment of the memory complexity.

Principle. Let $\mathbf{H}$ be the parity-check matrix of the code, which has n columns and $n-k$ rows, where n is the length of the code and k its dimension (the codewords forming its kernel). The goal of SDP is to find a vector $\mathbf{s}$ such that $\mathbf{Hs}$ has some prescribed Hamming weight.

Prange's algorithm selects at random a subset I of $n-k$ columns of $\mathbf{H}$, which defines a square submatrix $\mathbf{H}_I$, and inverts the subsystem defined by $\mathbf{H}_I$. If the nonzero coefficients of the vector solution to SDP correspond to columns of $\mathbf{H}$ that are all in $\mathbf{H}_I$, then inverting the subsystem defined by $\mathbf{H}_I$ will allow to retrieve this solution. And since $\mathbf{H}_I$ is square, Gaussian elimination can be used to invert the system, which will have few solutions on average. This procedure is repeated for random choices of columns I until a solution is found.

Quantum ISD. In the quantum setting, one can speedup this algorithm using Grover's quantum search, as proposed by Bernstein [11] (an algorithm that we will call "quantum Prange" in what follows). Indeed, the subsets of columns I form a well-defined search space, and solving the subsystem $\mathbf{H}_I$ allows to test if I is solution to our problem. Using Grover's search, the number of iterations decreases to a square root of its classical value. Despite this asymptotic speedup, we should note that in practice, the cost of solving $\mathbf{H}_I$, though a polynomial factor, is far from negligible.

Similarly to the classical setting, improved quantum ISD algorithms were introduced later on by Kachigar and Tillich [35] and Kirshanova [37] using various techniques from classical ISD, notably quantum verzsions of the MMT [40] and BJMM [6] algorithms using quantum walks. More recently, Kimura et al. [36] presented another trade-off between time and memory using a combination of Kirshanova's algorithm, Both and May's classical ISD algorithm [16], and Grover's algorithm. Chailloux et al. [20] extended the scope of quantum Prange by adapting it to other metrics than the Hamming one. Since we are particularly targeting the memory complexity of quantum ISD, and since we consider the Hamming

case only, the improvements of [35,37,36,20] will not be considered further in this paper.

Improving Quantum Prange. Esser et al. [26,27] introduced several improvements to quantum Prange, leading to polynomial runtime improvements: first, an optimization using the Lee-Brickell algorithm [38] and an optimization based on preprocessing the parity-check matrix to systematic form, which we will not reuse here.

In [27], motivated by the large number of qubits required, they proposed a hybrid quantum-classical trade-off. The idea is to guess a first part of the 0 coefficients' positions in a precomputation on a classical computer, and then to carry a smaller instance of the quantum Prange's algorithm. Another possible optimization, in the same fashion, is to not consider all of the equations in the square systems one aims at solving. The combination of these two optimizations reduces the qubit cost by a constant, but increases the time exponentially.

More recently, Perriello et al. [47] performed complete quantum cost estimates of quantum Prange, which we will compare to for the parameters of the NIST code-based candidates.

Memory Complexity and Comparisons. The parity-check matrix $\mathbf{H}$ itself, and its sub-matrices that one tries to invert, occupies a space $\mathcal{O}(n^2)$. Therefore the memory complexity of Prange's algorithm, classical or quantum, is not negligible. In code-based cryptosystems, n ranges between 10^3 and 10^5, leading to millions or billions of logical qubits. This is much more than the thousands required by Shor's algorithm for factoring RSA semiprimes [30] or even exhaustive search of AES keys [34], which is explicitly used by the NIST as a benchmark for post-quantum security levels [43].

Looking back at the optimizations of quantum ISD [26,27,47], the number of qubits could never decrease below $\mathcal{O}(n^2)$, despite improvements in the constants. This is simply because these quantum algorithms used Gaussian elimination to invert the sub-matrices considered during the Grover search iterations.

At the start of the post-quantum standardization process [43], the NIST focused on the metrics of gate count and depth. Specifically, they suggested to compare total gate counts under a limitation of circuit depth (i.e., total running time), denoted MAXDEPTH, which ranges from 2^{40} to 2^{96}. Under these metrics, the total number of qubits is not a limiting parameter, since one expects the algorithms to become massively parallel. However, the number of qubits that each individual machine uses remains of significance. Reducing this amount would allow more flexible trade-offs for such parallel searches.

Contribution and Organization. In this paper, we optimize the qubit count of quantum ISD. Our innovation lies in the way the sub-matrices considered throughout the quantum search are represented, and inverted.

We use Wiedemann's matrix inversion algorithm [54]. This technique, in our case, is interesting because it reduces the inversion of a dimension-n matrix to

computing a series of $\mathcal{O}(n)$ matrix-vector products, and using the Berlekamp-Massey algorithm on linear recurrent sequences of size $\mathcal{O}(n)$. We show how to implement the matrix-vector product by a sub-matrix $\mathbf{H}_I$ given a compact representation of the choice I of columns. Next, we give a quantum implementation of the Berkekamp-Massey algorithm over $\mathbb{F}_2$ using $\mathcal{O}(n)$ space only. This gives us a circuit for the Grover iteration in quantum Prange that uses $\mathcal{O}(n)$ qubits.

This first implementation is optimized for space, but the resulting gate count $\mathcal{O}\big(n^3\big)$ and circuit depth $\mathcal{O}\big(n^2 \log n\big)$ suffer from large constant factors. This motivates us to give another trade-off, by changing the representation of I in the algorithm. In the first version, I is represented as a *selection* of columns of the matrix $\mathbf{H}$, i.e., a vector of length n and weight $n-k$, similarly to what is commonly done in previous works. In the second version, I is represented as a permutation of the columns (an idea that appeared in [46], but without further details). This permutation is implemented using a *switching network* with $\mathcal{O}\big(n \log^2 n\big)$ switches. The qubit count increases to $\mathcal{O}\big(n \log^2 n\big)$, but this is compensated by a significant improvement in gate count.

First of all, we reduce the depth to $\mathcal{O}\big(n^2\big)$, becoming asymptotically optimal, and the Toffoli gate count to $\mathcal{O}\big(n^2 \log^2 n\big)$ (while the total gate count remains $\mathcal{O}\big(n^3\big)$), improving over circuits based on Gaussian elimination. This is particularly interesting from an implementation standpoint, as Toffoli gates, which contain nonlinearity, are considered much harder to implement than NOT and CNOT gates. This also shows that there are further advantages to Wiedemann inversion than just space. Finally, when the parity-check matrix is block-circulant, as in BIKE and HQC, we show that the *total* gate count of the Grover iteration can be reduced to $\mathcal{O}\big(n^2 \log^2 n\big)$ (Theorem 4), though our current implementation achieves only a minor gain, at the expense of circuit depth.

Inverting large sparse or implicit matrices is a building block of other quantum algorithms, notably quantum algorithms for solving multivariate quadratic systems [28,14]. The space complexity was not discussed in [28], and in [14], the generic Bennett-Tompa reversibility trade-off is used [7], which leads to additional non-negligible factors in time and space. Our results could lead to improvements in both cases, although this would require a dedicated analysis.

On a side note, it has been pointed out by a reviewer that our space optimization of quantum ISD is possible because the quantum circuit is *instance-dependent*, i.e., the parity-check matrix $\mathbf{H}$ is encoded in the circuit instead of being represented as an input. This is not an uncommon occurrence in quantum cryptanalysis: typical implementations of Shor's algorithm [30] use a precomputation which also makes the circuit instance-dependent. However, it might be interesting to develop further such techniques to reduce the qubit count.

Organization of the paper. Section 2 describes Prange's algorithm and Wiedemann's inversion [54]. In Section 3 we detail the quantum version of Prange's algorithm, and explain how to replace the linear algebra part with Wiedemann's algorithm. Our main result is stated here, but its proof spans the following sections.

First, in Section 4, we give our implementation of Wiedemann inversion, including a quantum reversible circuit for the Berlekamp-Massey algorithm [8]. This implementation uses as black-box a quantum algorithm for matrix-vector product. Then, in Section 5, we propose two such implementations. The first one (space-optimized) reduces the space to $\mathcal{O}(n)$. The second (Toffoli-optimized) offers a trade-off with a space $\mathcal{O}\left(n \log^2 n\right)$ and a reduced Toffoli cost. We give precise formulas for the costs of all these circuits. Finally, in Section 6, we evaluate these costs for the parameters of code-based cryptosystems.

We implemented the quantum circuits considered in this paper using the Qiskit framework [49]. Our code is available at `https://gitlab.inria.fr/capsule/quantum-isd-less-qubits`.

2 Preliminaries

Throughout this paper, n and $k \leq n$ are integers. We use bold notations for vectors $(\mathbf{e}, \mathbf{s} \ldots)$ and uppercase letters for matrices $(\mathbf{A}, \mathbf{H}, \ldots)$. The transpose of a matrix is denoted $\mathbf{A}^T$. Since we work only in $\mathbb{F}_2$, addition $+$ will always refer to addition in $\mathbb{F}_2$ (binary XOR) or in a vector space over $\mathbb{F}_2$. The symbol δ_{ij} corresponds to the Kronecker delta function, that equals 1 if and only if $i = j$, and 0 otherwise.

For q a prime, a *linear code* of length n and dimension k over $\mathbb{F}_q$ is a k-dimensional vector subspace $\mathbb{F}_q^n$, which can be defined as the kernel of a *parity-check* matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$. In this paper we consider the case of *binary* codes, i.e., $q = 2$. The Hamming weight hw is defined, for vectors of any length, by the number of non-zero coordinates. We consider the Syndrome Decoding problem $SD(n, k, w)$: given as input $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$ and $\mathbf{s} \in \mathbb{F}_2^{n-k}$ (the error syndrome), find $\mathbf{e} \in \mathbb{F}_2^n$ such that $\mathrm{hw}(\mathbf{e}) = w$.

While the decision version of this problem, i.e., the existence of such a solution, is well-known to be NP-complete [9], we consider the search version where $\mathbf{H}$ is sampled uniformly at random, and the existence of the solution is guaranteed. With proper choice of the parameters n, k, w, this case is still believed to be hard for classical and quantum computers alike.

Problem 1 (Random Decoding). Given as input $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$ sampled uniformly at random, and $\mathbf{s} = \mathbf{He}$ where $\mathbf{e}$ is sampled uniformly at random from vectors of weight w, find $\mathbf{e}$.

2.1 Prange's Algorithm

In [48], Prange introduced an algorithm for generic decoding (Algorithm 1) which forms the basis of the family of Information Set Decoding (ISD) algorithms. We use it to solve Problem 1. The idea of the algorithm is to select a random subset of the columns of $\mathbf{H}$. Let $S_{n,k}$ be the set of all subsets of $\{0, \ldots, n-1\}$ with $n - k$ elements.

Algorithm 1 Prange's algorithm, binary case [48].

Input: parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, syndrome $\mathbf{s} \in \mathbb{F}_2^{n-k}$, weight $w \leq n-k$
Output: vector $\mathbf{e}$ such that $\mathbf{He} = \mathbf{s}$ and $\mathrm{hw}(\mathbf{e}) = w$
repeat
  Choose a random subset $I \in S_{n,k}$
  If $\mathbf{H}_I$ is not invertible, **continue**
  Solve the linear system $\mathbf{H}_I\mathbf{e}_I = \mathbf{s}$ for $\mathbf{e}_I$
until $\mathrm{hw}(\mathbf{e}_I) = w$
Return $\mathbf{e}$ = extension of $\mathbf{e}_I$ with zeroes

Given $I \in S_{n,k}$, we define $\mathbf{H}_I$ as the sub-matrix of $\mathbf{H}$ which keeps these columns only. If all non-zero positions of the vector $\mathbf{e}$ are in I, the equation system $\mathbf{H}_I\mathbf{x} = \mathbf{s}$ admits a solution $\mathbf{x}$ of weight w, and extending this solution by zeroes provides a solution to the SDP[1].

Runtime. As a first remark, it is well known that a random square matrix in $\mathbb{F}_2$ is invertible with probability at least 0.288 [21]. In order to bound easily the runtime of Prange's algorithm, we make the following heuristic assumption.

Heuristic 1. A matrix $\mathbf{H}_I$, where I leads to the solution, is invertible with probability 0.288.

Intuitively, the invertibility of the sub-matrix $\mathbf{H}_I$ should be independent from whether I is a solution or not. While we cannot exclude pathological values of $\mathbf{H}$ and $\mathbf{s}$, we can assume that such cases occur with negligible probability.

Theorem 1. *Under Heuristic 1, Algorithm 1 succeeds with $\frac{1}{p}$ iterations on average, where:*

$$p = 0.288\frac{\binom{n-k}{w}}{\binom{n}{w}} \quad , \tag{1}$$

and has a time complexity $\mathcal{O}(\frac{1}{p}(n-k)^\omega)$ where ω is the matrix multiplication exponent.

Proof. A first important remark is that the algorithm does succeed: this is because there are in general *many* solutions I, and by Heuristic 1, one of them will lead to an invertible sub-matrix – so it will eventually be found.

Consider the columns $H_{i_1}, \ldots, H_{i_w}$ of $\mathbf{H}$ that correspond to the non-zero coefficients of the solution $\mathbf{e}$. The loop succeeds whenever $\{i_1, \ldots, i_w\} \subseteq I$ and $\mathbf{H}_I$ is invertible. By Heuristic 1, the number of good choices for I is: $0.288\binom{n-w}{n-k-w}$, while the total number of possible I is $\binom{n}{n-k}$. This gives a probability of success in the loop $p = 0.288\frac{\binom{n-w}{n-k-w}}{\binom{n}{n-k}} = 0.288\frac{\binom{n-k}{w}}{\binom{n}{w}}$.

[1] Note that we have considered here the binary case only; the case of a generic q requires more care.

The average number of iterations before achieving a success follows from this. Each iteration requires to invert a linear system of dimension $n-k$, which requires $\mathcal{O}((n-k)^\omega)$ elementary operations. □

The subsequent improvements to Prange's ISD [53,38,40,6,41,16,32,25] put less constraints on I, which decreases the number of loop iterations; however they increase the complexity of recovering $\mathbf{e}$ during the iteration. Importantly, any improvement in the time complexity exponent comes at the expense of a larger memory complexity, which is why we do not consider these variants in what follows.

2.2 Wiedemann's Inversion Algorithm

This section follows Wiedemann [54].

Finding Relations. The Berlekamp-Massey algorithm [8,39] takes as input a sequence $(s_0, \ldots, s_{N-1}) \in \mathbb{F}_q^N$ satisfying a linear recurrence relation over $\mathbb{F}_q$:

$$\exists \ell \leq N, \exists c_0 \neq 0, c_1, \ldots, c_\ell, \forall i \leq N, c_0 s_i + c_1 s_{i-1} + \ldots + c_\ell s_{i-\ell} = 0 \ . \tag{2}$$

This relation is represented by a polynomial $C(X) := c_0 + c_1 X + \ldots + c_\ell X^\ell \in \mathbb{F}_q[X]$, of degree ℓ. Assuming that $2\ell \leq N$, the Berlekamp-Massey algorithm returns such a polynomial $C(X)$ with the smallest possible degree. We defer the details of this algorithm to Section 4.2. It runs in $\mathcal{O}(N^2)$ operations, using $\mathcal{O}(N)$ space. Using its similarity to an extended Euclidean algorithm on polynomials [24], this time can be reduced to $\tilde{\mathcal{O}}(N)$, but this will not be important for us as the time complexity of Berlekamp-Massey will not be dominant in our algorithms.

Wiedemann's Algorithm. Wiedemann [54] considers the problem to invert a sparse matrix over $\mathbb{F}_q$, or more generally, any matrix $\mathbf{A}$ for which only a black-box matrix-vector product is provided. In the case of sparse matrices, this is motivated by the efficiency of such products, and the low space complexity.

Given inputs $\mathbf{A}$ and $\mathbf{s} \in \mathbb{F}_q^n$, the goal is to find the unique $\mathbf{x}$ such that $\mathbf{A}\mathbf{x} = \mathbf{s}$. In the following, we assume that $\mathbf{A}$ is invertible. Wiedemann [54] provides further analysis to deal with non-invertible matrices, but this will not be necessary for us, as we will essentially need to succeed with constant probability for random matrices (therefore, succeeding for invertible matrices is sufficient for us).

Let S be the space spanned by $\{\mathbf{A}^i\mathbf{s}, i \in \mathbb{N}\}$ where $\mathbf{A}^0 = \mathbf{I}$ is the identity matrix. We consider the action of $\mathbf{A}$ on this space, defined by an operator A_S with minimal polynomial $P(X) \in \mathbb{F}_q[X]$. The polynomial P is normalized to have its first coefficient equal to 1. Let $Q(X) = (1 - P(X))/X \in \mathbb{F}_q[X]$, which is of degree $n-1$ at most. We have:

$$P(\mathbf{A})\mathbf{s} = \mathbf{0} \implies \mathbf{A}\left(Q(\mathbf{A})\mathbf{s}\right) = \mathbf{s} \implies \mathbf{x} = Q(\mathbf{A})\mathbf{s} \ . \tag{3}$$

Given Q, evaluating $Q(\mathbf{A})\mathbf{s}$ can be done by a series of n matrix-vector products and $\mathcal{O}(n)$ temporary space, by Horner's method. Therefore, the search for $\mathbf{x}$ is reduced to the search for P.

The search for P can be reduced to finding linear recurrences, as follows. Since evaluating $\mathbf{A}^i\mathbf{s}$ yields a sequence of vectors in $\mathbb{F}_q^n$, and not of scalars, one selects a (random) vector $\mathbf{u} \in \mathbb{F}_q^n$ and computes the sequence of projections: $\left(\mathbf{u}^T\mathbf{A}^i\mathbf{s}, i \in \mathbb{N}\right)$. This sequence satisfies a linear recurrence, with a minimal polynomial $C(X)$ that divides $P(X)$. Since P is of degree n, only $2n$ terms need to be computed. In fact, after on expectation $\mathcal{O}(\log n)$ tries with random vectors $\mathbf{u}$, one will obtain $C(X)$. But it is possible to arrive at this result faster using a slightly more technical algorithm which finds first a divisor C_0 of P, then reduces the problem to finding P/C_0, etc. This is summarized in Algorithm 2.

Algorithm 2 Wiedemann's algorithm for inversion.

Input: invertible matrix $\mathbf{A}$ accessed only by a black-box product operator: $\mathbf{y} \mapsto \mathbf{Ay}$; vector $\mathbf{s}$
Output: $\mathbf{x} = \mathbf{A}^{-1}\mathbf{s}$

$\mathbf{t} \leftarrow \mathbf{s}$, $\mathbf{y} \leftarrow 0$
$d \leftarrow 0$ ▷ Current degree of the polynomial
repeat
  Select $\mathbf{u}$ uniformly at random
  Compute the first $2(n-d)$ terms of the sequence $\mathbf{u}^T\mathbf{A}^i\mathbf{t}$
  Compute $C(X)$, the minimal polynomial of this sequence
  Let $C'(X) = (C(X) - 1)/X$
  $\mathbf{y} \leftarrow \mathbf{y} + C'(\mathbf{A})t$
  $\mathbf{t} = \mathbf{s} + \mathbf{Ay}$
  $d \leftarrow d + \deg(C)$
until $\mathbf{t} = 0$
Return $-\mathbf{y}$

The probability of success of this algorithm follows Lemma 1.

Lemma 1 (From [54], Section VI). *For $k > 1$, the probability that after k iterations in the main loop of Algorithm 2, one has $\mathbf{t} = 0$ (and $\mathbf{A}(-\mathbf{y}) = \mathbf{s}$), is lower bounded by:*

$$1 - \log\left(\frac{q^{k-1}}{q^{k-1}-1}\right) \quad . \tag{4}$$

As a constant success probability will be enough for us, we can run Algorithm 2 with a constant number of loops. In particular with $q = 2$, by using $k = 2$ we ensure a probability of success bigger than $2^{-1.70}$. As proposed by Wiedemann, we will also replace the selected vectors $\mathbf{u}$ by deterministic unit vectors, which consist merely in selecting the first and second coordinates of $\mathbf{A}^i\mathbf{s}$ (though it would not be much more difficult to take random vectors).

It should be noted that Wiedemann's algorithm appeared previously in a quantum context in the algorithms of [28,14] for multivariate quadratic equation

systems. These algorithms construct large sparse matrices (Macaulay matrices) which need to be inverted. However, neither of these works considered a full reversible implementation of Wiedemann's algorithm; instead they used generic reversibilisation results. In [28] they noticed that the algorithm could be implemented in a naive way, increasing the space complexity. In [14] they used the generic Bennett-Tompa trade-off [7] which introduces subexponential factors in the complexity (which disappear in the asymptotic complexity estimates).

In order to get a satisfying space complexity for the quantum Prange's algorithm, we will need to implement Wiedemann's algorithm in a reversible *and space-efficient* way. This is the goal of Section 4.

3 Quantum Preliminaries

In this section we give the required preliminaries of quantum computing and quantum ISD, including the formulation of our main result (Theorem 3) which relies on all the building blocks studied in the remainder of the paper.

We refer to [42] for an introduction to the notions of quantum computing (quantum states, amplitudes, ket notation $|\cdot\rangle$). We describe quantum algorithms in the *quantum circuit model* as a sequence of *quantum gates* applied to a set of qubits. We stand only at the *logical* level, in which gates can be freely applied to any qubit or pair of qubits without inducing any error. *Ancilla qubits* are those which start in the state $|0\rangle$ and are restored to $|0\rangle$ after applying the circuit.

Many advanced quantum cryptanalysis algorithms make use of quantum-accessible memories (also known as qRAM) [35,20]. qRAM can be seen as an abstraction of quantum hardware in which writing and / or reading in quantum superposition from a large-scale memory could be an efficient operation. It is notably used in certain quantum walk algorithms which require to maintain and update a large superposed memory state. However, it is almost certain that near-term quantum devices will not benefit from such capabilities. Going back to the *baseline* quantum circuit model, in which only fixed-arity gates can be used, these advanced algorithms lose their advantage. We refer to the aforementioned papers for more details on the qRAM model.

In this paper, we are interested in making conservative hardware assumptions, in which qRAM is not available and the number of logical qubits is limited. The new circuits that we design are entirely classical reversible circuits, which contain only NOT (X), CNOT (controlled-X, or CX) and Toffoli (double-controlled X, or CCX) gates. The other quantum gates required to run quantum ISD algorithms are Hadamard gates (H) used in the iterations of Quantum Amplitude Amplification, and rotation gates used in the construction of Dicke states [4] (which we define later on).

From an implementation perspective, CCX gates are known to be much more costly than X and CX gates, which is why they form the main target for optimization (e.g., in [30]).

3.1 Quantum Search

Grover's quantum search algorithm [31] provides a quadratic speedup for any exhaustive search problem, which can be defined as the search of a preimage of 1 of a function $f : \{0,1\}^n \to \{0,1\}$, where $\{0,1\}^n$ is the search space and f distinguishes "good" elements ($f(x) = 1$) from "bad" ones ($f(x) = 0$).

Prange's information set decoding algorithm can be rephrased as such a search problem, which is why one can use quantum search here [11]. However, for this precise context it is better to rely on the generalization of Grover's search known as Quantum Amplitude Amplification (QAA) [17]. QAA can start from any probabilistic algorithm (implemented as a quantum circuit) that succeeds with probability p, and needs $\mathcal{O}\left(1/\sqrt{p}\right)$ iterations. Furthermore, it is quite robust if the probability of success is not known exactly, and p is only a lower bound. In that case an adapted procedure still succeeds in time $\mathcal{O}\left(1/\sqrt{p}\right)$.

For the specific case of quantum ISD, we apply QAA in the following way.

Theorem 2 (Consequence of Theorem 2 and Theorem 3 of [17]). *Let U be a quantum circuit that, on input $|0\rangle$, produces a uniform superposition of N basis states (a subset $X \subseteq \{0,1\}^n$):*

$$U|0\rangle = \frac{1}{\sqrt{N}} \sum_{x \in X} |x\rangle \quad . \tag{5}$$

Let O_f be a quantum circuit that realizes a phase oracle *for a function $f : X \mapsto \{0,1\}$, where $|f^{-1}(1)| = M$:*

$$\forall x \in X, O_f|x\rangle = (-1)^{f(x)}|x\rangle \quad . \tag{6}$$

Then, there exists a quantum algorithm that outputs an $x \in f^{-1}(1)$, and makes an expected number of $\Theta(\sqrt{N/M})$ calls to O_f and U.

More precisely, QAA is a procedure that runs with a fixed number of *iterations*, which repeat the *setup* operation U and the *test* operation O_f. A QAA iteration is the unitary $Q = -UO_0U^{-1}O_f$, where U and O_f are defined in Theorem 2, and O_0 is the unitary defined by $O_0|x\rangle = (-1)^{\delta_{0x}}|x\rangle$, where δ_{0x} is the Kronecker delta. That is, it flips the phase iff $x = 0$. One should note that O_0 is essentially an n-bit multi-controlled Z gate, which is equivalent (up to a Hadamard transform) to a multi-controlled Toffoli gate, which can be implemented with $\mathcal{O}(n)$ Toffoli gates [42]. As soon as U and / or O_f use more than $\mathcal{O}(n)$ depth, qubits and gates, the cost of this operation becomes negligible.

Let $\theta := \arcsin\sqrt{\frac{M}{N}}$. It can be shown [17] that starting from $U|0\rangle$ and applying k iterations of QAA, one produces the state:

$$Q^kU|0\rangle = \left(\frac{\sin\left((2k+1)\theta\right)}{\sqrt{M}} \sum_{x \in f^{-1}(1)} |x\rangle\right) + \left(\frac{\cos\left((2k+1)\theta\right)}{\sqrt{N-M}} \sum_{x \in f^{-1}(0)} |x\rangle\right) . \tag{7}$$

This is why, knowing M (hence θ) in advance, we can succeed with probability close to 1 by setting $k = \left\lfloor \frac{\pi}{4\theta} \right\rfloor$. If we have only upper and lower bounds on M, we can use the following lemma.

Lemma 2. *Assume that $M_\ell \leq M \leq M_u$. Run $k = \left\lfloor \frac{\pi}{4 \arcsin(\sqrt{M_u/N})} - \frac{1}{2} \right\rfloor$ iterations of QAA. The probability of success is:*

$$p_{\text{succ}} \geq \sin^2 \left(\frac{\pi}{2} \frac{\arcsin \sqrt{M_\ell/N}}{\arcsin \sqrt{M_u/N}} - 2 \arcsin \sqrt{M_\ell/N} \right) \quad . \tag{8}$$

Proof. The choice of k ensures that $(2k+1) \arcsin \sqrt{\frac{M}{N}} \leq (2k+1) \arcsin \sqrt{\frac{M_u}{N}} \leq \frac{\pi}{2}$, which means the sin remains an increasing function. We can use Equation 7 to bound the probability of measuring a good x:

$$\begin{aligned} p_{\text{succ}} &:= \sin^2 \left((2k+1) \arcsin \sqrt{\frac{M}{N}} \right) \geq \sin^2 \left((2k+1) \arcsin \sqrt{\frac{M_\ell}{N}} \right) \\ &\geq \sin^2 \left(\left(\frac{\pi}{2 \arcsin(\sqrt{M_u/N})} - 2 \right) \arcsin \sqrt{\frac{M_\ell}{N}} \right) \quad . \end{aligned}$$

□

In particular, if M_ℓ and M_u are very close to M, then the success probability will become negligibly close to 1 (as it is when M is known exactly). If they are close up to a constant factor, then we ensure a constant probability of success.

3.2 Quantum ISD

Bernstein [11] noticed that Algorithm 1 is an exhaustive search for which one can use Grover's algorithm. Quantum ISD was subsequently improved in [35], but not in a way that can be useful for us, since we refrain from using quantum RAM and exponential space.

Adapting Prange's algorithm to the QAA framework is easily done, by defining the operators U and O_f:

- U produces a uniform superposition of subsets $I \subseteq \{0, \ldots, n-1\}$ of size $n-k$:
$$|\mathcal{I}\rangle = \frac{1}{\sqrt{|S_{n,k}|}} \sum_{I \in S_{n,k}} |I\rangle \quad , \tag{9}$$
where I is simply represented as a bit-vector of length n, where "1" in position i indicates that $i \in I$.
Such a quantum state is known in the literature as a *Dicke state*, and several efficient methods exist to compute it [4]. The cost of these methods is always significantly smaller than the cost of linear algebra in O_f (see, e.g., [47]).
- O_f is an oracle for the function f that takes as input I, and returns 1 if and only if $\mathbf{H}_I$ is invertible and $\mathbf{H}_I^{-1}\mathbf{s}$ is of Hamming weight w.
Bernstein estimated that the evaluation of f would require $\mathcal{O}(n^3)$ "bit operations" [11]. This analysis was refined by further works [47]. However, to date, all implementations of O_f start by writing the matrix $\mathbf{H}_I$, then inverting it using Gaussian elimination. This strategy obviously requires at least $(n-k)^2$ qubits.

Decoding One out of Many. It is known that ISD algorithms can gain a speedup if the adversary's goal is to decode a single syndrome out of many (the so-called DOOM problem [51]). A specific case of DOOM happens in the quasi-cyclic variant of the problem, which is used in BIKE [3] and HQC [1]. Indeed, in that case, one has $n = 2k$ and the parity-check matrix $\mathbf{H}$ is formed of two circulant blocks $\mathbf{H}_1$ and $\mathbf{H}_2$. If $\mathbf{e} = (\mathbf{e}_1\mathbf{e}_2)$ is the error vector, one has $(\mathbf{H}_1\mathbf{H}_2)(\mathbf{e}_1\mathbf{e}_2) = \mathbf{s}$. Let R_i be the rotation of a vector's coordinates by i positions left, then due to the circulant structure of the blocks, one has:

$$\forall 0 \leq i < k, (\mathbf{H}_1\mathbf{H}_2)(R_i(\mathbf{e}_1)R_i(\mathbf{e}_2)) = R_i(\mathbf{s}) \ . \tag{10}$$

As a consequence, $\mathbf{e}$ can be found by decoding *any* of the k syndromes $R_i(\mathbf{s})$. One can then adapt Prange's algorithm as follows. When a set I has been chosen, instead of computing $\mathbf{H}_I^{-1}\mathbf{s}$ and checking the Hamming weight of the result, one forms the $k \times k$-dimensional matrix whose i-th column is $R_i(\mathbf{s})$, computes $\mathbf{H}_I^{-1}(R_0(\mathbf{s}) \cdots R_{k-1}(\mathbf{s}))$ and looks for a column of weight w. This increases the probability of success by a factor k, and reduces the number of iterates in quantum Prange by a factor $\sqrt{k}$. This use of DOOM was discussed in [47].

Intriguingly, we do not know how to use DOOM in the context of Wiedemann inversion, because Wiedemann inverts the matrix on a single given vector. Doing this for another vector essentially requires to re-run the whole algorithm, without any gain. Therefore we will lose a factor $\sqrt{k}$ in time complexity compared to [47] for the cases of BIKE and HQC.

3.3 Quantum Prange Using Wiedemann Inversion

We give a very abstract formulation of our main result, where the matrix is only accessed via a black-box representation of I and $\mathbf{H}_I$. In particular, this allows to consider alternative ways to represent the selection of a subset of columns.

From now on, we let $\mathcal{J}$ be a set of bit-strings of fixed size such that there exists a surjective mapping F from $\mathcal{J}$ to $S_{n,k}$, and furthermore, each subset has the same number of preimages. Consequently, sampling uniformly at random from $\mathcal{J}$ allows to sample uniformly at random from $S_{n,k}$, even though $\mathcal{J}$ may be a bigger set. Furthermore, given any $J \in \mathcal{J}$, we can extend our notation for sub-matrices by writing $\mathbf{H}_J$ instead of $\mathbf{H}_{F(J)}$.

For our main result, we need a stronger heuristic than Heuristic 1, which indicates that being a solution, being an invertible matrix, and being a matrix on which Algorithm 2 succeeds with two iterations, are roughly independent events.

Heuristic 2. The proportion of matrices $\mathbf{H}_I$ where I is a solution, which are invertible, and which Algorithm 2 can invert with $k = 2$ on input $\mathbf{s}$, is at least $0.288 \times 2^{-1.70} \simeq 2^{-3.50}$.

We now assume that we have the following:

- A quantum circuit Init that, on input $|0\rangle$, returns $|\mathcal{J}\rangle = \frac{1}{\sqrt{|\mathcal{J}|}} \sum_{J \in \mathcal{J}} |J\rangle$

- A quantum circuit $\mathsf{Mult}_{\mathbf{H}}$ that, on input $|J\rangle|\mathbf{x}\rangle|\mathbf{y}\rangle$, where $\mathbf{x},\mathbf{y} \in \mathbb{F}_2^{n-k}$, returns $|J\rangle|\mathbf{y}+\mathbf{H}_J\mathbf{x}\rangle|\mathbf{x}\rangle$

By their "space" complexity, we shall mean their entire qubit count, including ancillas. Our main result, building upon our implementation of Wiedemann's algorithm in the quantum setting, integrates these two components in quantum ISD.

Theorem 3. *Given a circuit for* Init *with* $S(\mathsf{Init})$ *qubits and* $G(\mathsf{Init})$ *gates, and a circuit for* $\mathsf{Mult}_{\mathbf{H}}$ *with* $S(\mathsf{Mult}_{\mathbf{H}})$ *qubits and* $G(\mathsf{Mult}_{\mathbf{H}})$ *gates, under Heuristic 2, there exists a quantum algorithm that solves the SDP with constant probability, using space:* $\max(S(\mathsf{Init}), S(\mathsf{Mult}_{\mathbf{H}}) + \mathcal{O}(n))$*, and gates:*

$$\mathcal{O}\left(\sqrt{\frac{\binom{n}{w}}{\binom{n-k}{w}}} \times \left(G(\mathsf{Init}) + (n-k)G(\mathsf{Mult}_{\mathbf{H}}) + (n-k)^2\right)\right) \quad . \tag{11}$$

Proof. The algorithm is simply an adaptation of quantum Prange using QAA. Formally, our goal is not exactly to recover a subset I that yields the error vector $\mathbf{e}$, but a representation of it through J.

The operator U is simply Init. For the operator O_f, we use our quantum implementation of Wiedemann's algorithm (Lemma 3), which has gate count $\mathcal{O}\big((n-k)G(\mathsf{Mult}_{\mathbf{H}}) + (n-k)^2\big)$ and uses $S(\mathsf{Mult}_{\mathbf{H}}) + \mathcal{O}(n)$ space. Importantly, in case of failure in Wiedemann's algorithm, f will return 0. In case of success, we obtain the vector $\mathbf{x}$ such that $\mathbf{H}_J\mathbf{x} = \mathbf{s}$. It remains to test if its Hamming weight is equal to w: the cost of this step is negligible with respect to the other components of the algorithm.

The number of iterations to perform depends on the probability that a given $J \in \mathcal{J}$ satisfies f, i.e., that the corresponding subset is solution, that $\mathbf{H}_J$ is an invertible matrix, and that Wiedemann's algorithm with two iterations succeeds. Under Heuristic 2, this probability can be lower bounded by: $2^{-3.50}\frac{\binom{n-k}{w}}{\binom{n}{w}}$. The result follows from Lemma 2. □

3.4 Quantum Circuit Components

In order to implement the Berlekamp-Massey and Wiedemann's algorithms in an efficient and reversible manner, we need quantum circuits for several basic operations. These circuits are folklore and / or simple and / or borrowed from previous works; they are constructed entirely from X, CX and CCX gates.

We summarize here the main results needed, and the interested reader can find more details in Section A.

Fan-in. A *fan-in* circuit implements the operation:

$$|v_0, \ldots, v_{n-1}, b\rangle \mapsto |v_0, \ldots, v_{n-1}, b + (\sum_i v_i)\rangle \quad .$$

It can be done using $\mathcal{O}(n)$ CX gates and in depth $\mathcal{O}(\log n)$.

Fan-out. A *fan-out* circuit implements the operation: $|b, 0, \ldots, 0\rangle \mapsto |b, b, \ldots, b\rangle$. It can be done using n CX gates and depth $\mathcal{O}(\log n)$.

Controlled-shift. A *controlled-shift* by a constant k maps:

$$\begin{cases} |0, v_0, \ldots, v_{n-1}\rangle \mapsto |0, v_0, \ldots, v_{n-1}\rangle \\ |1, v_0, \ldots, v_{n-1}\rangle \mapsto |1, v_k, \ldots, v_{n-1}, v_0, \ldots, v_{k-1}\rangle \end{cases} \tag{12}$$

The shift is controlled by the first qubit. It can be done using $4n$ CX gates, $3n$ CCX gates, $\mathcal{O}(n)$ ancilla qubits and $\mathcal{O}(\log n)$ depth.

Reversion. A *reversion* circuit maps a register of n bits of the form:

$$1, b_0, \ldots, b_{d-1}, 1, 0, \ldots, 0 \quad \text{to:} \quad 1, b_{d-1}, \ldots, b_0, 1, 0, \ldots, 0 \ ,$$

i.e., it reverts the order of bits without taking into account the trailing zeroes (d being a variable), and assuming that the first bit is 1. It can be done using $\mathcal{O}(n \log n)$ gates, depth $\mathcal{O}(n)$ and using $\mathcal{O}(n)$ ancilla qubits.

(Constant) Matrix-vector Multiplication. The multiplication of an n-bit vector $\mathbf{x} \in \mathbb{F}_2^n$ by a *constant* matrix $\mathbf{H} \in \mathbb{F}_2^{m \times n}$:

$$|\mathbf{x}\rangle\,|\mathbf{y}\rangle \xmapsto{\mathsf{MultConstant}_{\mathbf{H}}} |\mathbf{x}\rangle\,|\mathbf{y} + \mathbf{H}\mathbf{x}\rangle \ ,$$

can be implemented using $\leq mn$ CX gates (the exact number depends on the matrix $\mathbf{H}$), depth $\max(m, n)$ and no ancilla qubits.

Circulant Matrix-vector Multiplication. In Section 5.3 we will use a quantum circuit for multiplication of a vector by a constant circulant matrix: we borrow its principle from Gidney [29].

When $\mathbf{H}$ is a circulant matrix of dimension $n \times n$, there exists an implementation for $\mathsf{MultConstant}_{\mathbf{H}}$ using $\mathcal{O}\left(n^{\log_2 3}\right)$ CX gates, depth $\mathcal{O}\left(n^{\log_2 3}\right)$ and $\mathcal{O}(n)$ ancilla qubits.

4 Space-Optimized Reversible Wiedemann Inversion

In this section, we detail our reversible implementation of Wiedemann's matrix inversion, assuming that both the representation of column subsets (via the set $\mathcal{J}$) and the matrix-vector product are given as black-boxes, i.e., quantum circuits:

$$\begin{cases} \mathsf{Init} \ : |0\rangle \mapsto \frac{1}{\sqrt{|\mathcal{J}|}} \sum_{J \in \mathcal{J}} |J\rangle \\ \mathsf{Mult}_{\mathbf{H}} \ : |J\rangle\,|\mathbf{x}\rangle\,|\mathbf{y}\rangle \mapsto |J\rangle\,|\mathbf{y} + \mathbf{H}_J\mathbf{x}\rangle\,|\mathbf{x}\rangle \ . \end{cases} \tag{13}$$

We emphasize that the implementation of both components is not trivial, and that the time and space complexities of the iteration in Grover's search depend in majority on them. But these implementations are deferred to Section 5.

Algorithm 3 Wiedemann's algorithm, simplified.

Constant: matrix $\mathbf{H}$, $\mathbf{s}$
Input: J
Output: a Boolean $\mathsf{Success}$, and if $\mathsf{Success} = \text{True}$, a vector $\mathbf{x}$ such that $\mathbf{H}_J\mathbf{x} = \mathbf{s}$

1: $\mathbf{t} \leftarrow \mathbf{s}, \mathbf{y} \leftarrow 0, \mathbf{u} \leftarrow (1, 0, \ldots, 0)$
2: Compute the sequence $S = (\mathbf{u}^T\mathbf{H}_J^i\mathbf{t})_{0 \le i \le 2(n-k)}$ ▷ See Lemma 4
3: Compute the minimal polynomial $C(X)$ of the sequence using the Berlekamp-Massey algorithm ▷ See Algorithm 5
4: Let $C'(X) = (C(X) + 1)/X$
5: $\mathbf{y} \leftarrow \mathbf{y} + C'(\mathbf{H}_J)\mathbf{t}$ ▷ See Lemma 5
6: $\mathbf{t} \leftarrow \mathbf{t} + \mathbf{H}_J\mathbf{y}$ ▷ If success at the first step, here $\mathbf{t} = 0$
7: $\mathbf{u} \leftarrow (0, 1, 0, \ldots, 0)$
8: Compute the sequence $S = (\mathbf{u}^T\mathbf{H}_J^i\mathbf{t})_{0 \le i \le 2(n-k)}$
9: Compute the minimal polynomial $C(X)$ of the sequence
10: Compute $C'(X) = (C(X) + 1)/X$
11: $\mathbf{y} \leftarrow \mathbf{y} + C'(\mathbf{H}_J)\mathbf{t}$
12: If $\mathbf{H}_J\mathbf{y} = \mathbf{s}$, then set $\mathsf{Success}$ to True (False otherwise)
13: **Return** $\mathsf{Success}$, $\mathbf{y}$

The algorithm that we implement in this section is Algorithm 3, which corresponds to Wiedemann's algorithm with two loop iterations. Notice that in case we succeed in the first iteration, we will have $\mathbf{t} = 0$ at Step 8, so the output value $\mathbf{y}$ will remain unchanged. Therefore, whether the first or second loop iteration succeeds, the algorithm succeeds. Otherwise, even if the matrix is actually invertible, the Boolean flag $\mathsf{Success}$ will be set to False. This entire section proves the following result.

Lemma 3. *There exists a (classical) reversible circuit implementation of Algorithm 3 which uses* $S(\mathsf{Mult}_{\mathbf{H}}) + \mathcal{O}(n)$ *qubits and* $\mathcal{O}\big((n-k)G(\mathsf{Mult}_{\mathbf{H}}) + (n-k)^2\big)$ *gates.*

Proof. The remainder of this section proves that all steps of this algorithm can be implemented reversibly and efficiently.

- The computation of each sequence $(\mathbf{u}^T\mathbf{H}_J^i\mathbf{t})$ is done with Lemma 4.
- The evaluation of polynomials is done with Lemma 5.
- The Berlekamp-Massey algorithm for a sequence of length $\mathcal{O}(n-k)$ can be implemented with $\mathcal{O}\big((n-k)^2\big)$ gates and $\mathcal{O}((n-k)\log(n-k))$ depth by Lemma 6.

All these individual steps occupy a total of $\mathcal{O}(n)$ space for their outputs, which is erased by uncomputing them backwards once we have obtained the result. □

4.1 Evaluation of Matrix Powers

We elaborate here on a sequence of orthogonal polynomials in $\mathbb{F}_2[X]$. These polynomials arise from the fact that our $\mathsf{Mult}_{\mathbf{H}}$ circuit, i.e., our matrix multiplication, is performed *out of place*.

Indeed, if a matrix $\mathbf{A}$ is invertible, the operation $\mathbf{y} \mapsto \mathbf{A}\mathbf{y}$ is reversible. This means that there exists a reversible circuit performing the computation *in-place*: $|\mathbf{y}\rangle \mapsto |\mathbf{A}\mathbf{y}\rangle$. However, the reverse of this circuit would implement $|\mathbf{y}\rangle \mapsto |\mathbf{A}^{-1}\mathbf{y}\rangle$. This means that if we knew how to multiply by $\mathbf{A}$ in place, we would essentially also know how to invert $\mathbf{A}$.

This is the reason why we start from an *out-of-place* matrix-vector multiplication, which is much easier to implement: $|\mathbf{x}\rangle|\mathbf{y}\rangle \xmapsto{\mathsf{Mult}_\mathbf{A}} |\mathbf{y} + \mathbf{A}\mathbf{x}\rangle|\mathbf{x}\rangle$, where $\mathbf{x}, \mathbf{y}$ are two vectors. This is essentially a round of a Feistel scheme. It is easy to notice that $\mathsf{Mult}_\mathbf{A}$ is a self-inverse operation followed by a swap, so it is reversible.

Unfortunately, Wiedemann's algorithm requires to compute iterations $\mathbf{A}^i\mathbf{x}$ given a starting vector $\mathbf{x}$. Since we are computing $\mathsf{Mult}_\mathbf{A}$ out of place, naively computing a sequence of length $\mathcal{O}(n)$ would have us store $\mathcal{O}(n)$ intermediate vectors. We can do better than this through a family of orthogonal polynomials, which arise naturally by iterating $\mathsf{Mult}_\mathbf{A}$.

Polynomials. We define the following polynomials:

$$\begin{cases} P_{-1}(X) = 0, P_0(X) = 1 \\ \forall i \geq 1, P_i(X) = P_{i-2}(X) + XP_{i-1}(X) \end{cases} \tag{14}$$

Then, the circuit resulting of iterating i times $\mathsf{Mult}_\mathbf{A}$, noted ${\mathsf{Mult}_\mathbf{A}}^i$, is such that:

$$\forall i \geq 0, {\mathsf{Mult}_\mathbf{A}}^i |\mathbf{t}, \mathbf{0}\rangle = |P_i(\mathbf{A})\mathbf{t}, P_{i-1}(\mathbf{A})\mathbf{t}\rangle \quad . \tag{15}$$

The proof is an elementary induction over i. Indeed, for all i:

$$\begin{aligned} {\mathsf{Mult}_\mathbf{A}}^{i+1} |\mathbf{t}, \mathbf{0}\rangle &= \mathsf{Mult}_\mathbf{A}({\mathsf{Mult}_\mathbf{A}}^i |\mathbf{t}, \mathbf{0}\rangle) = \mathsf{Mult}_\mathbf{A} |P_{i+1}(\mathbf{A})\mathbf{t}, P_i(\mathbf{A})\mathbf{t}\rangle \\ &= |P_i(\mathbf{A})\mathbf{t} + \mathbf{A}P_{i+1}(\mathbf{A})\mathbf{t}, P_{i+1}(\mathbf{A})\mathbf{t}\rangle = |P_{i+2}(\mathbf{A})\mathbf{t}, P_{i+1}(\mathbf{A})\mathbf{t}\rangle \quad . \end{aligned} \tag{16}$$

It can be noticed that for all $i \geq 0$, P_i is of degree i. As a consequence, there exists a binary, lower triangular (invertible) matrix M_ℓ such that:

$$M_\ell \left(P_0(X), P_1(X), \ldots, P_\ell(X)\right)^T = \left(1, X, \ldots, X^\ell\right)^T \quad . \tag{17}$$

We can now explain how to perform two important steps in Wiedemann's algorithm:

- Evaluating a sequence $u^T\mathbf{A}^i\mathbf{t}$
- Evaluating $C(\mathbf{A})\mathbf{t}$ for a polynomial C

both reversibly, and using only linear additional space.

Lemma 4. *Let $\mathbf{A}$ be a matrix of dimension $n - k$. Given an implementation of $\mathsf{Mult}_\mathbf{A}$ with G gates, depth D and $\mathcal{O}(n-k)$ space, there exists a reversible circuit to compute the sequence $u^T\mathbf{A}^i\mathbf{t}$ for $i = 0, \ldots, \ell$ using $\mathcal{O}\left(\ell G + \ell^2\right)$ gates, depth $\mathcal{O}(\ell D + \ell \log \ell)$ and $\mathcal{O}(n - k + \ell)$ space.*

Proof. The idea is the following: we compute the $u^T \mathbf{A}^i \mathbf{t}$ as:

$$\left(u^T \mathbf{A}^0 \mathbf{t}, u^T \mathbf{A}^1 \mathbf{t}, \ldots, u^T \mathbf{A}^\ell \mathbf{t}\right)^T = M_\ell \left(u^T P_0(\mathbf{A})\mathbf{t}, u^T P_1(\mathbf{A})\mathbf{t}, \ldots, u^T P_\ell(\mathbf{A})\mathbf{t}\right)^T . \quad (18)$$

So, we only maintain two $(n-k)$-qubit registers for computing the successive $P_i(\mathbf{A})\mathbf{t}$ in place, and ℓ qubits for the sequence. Each time we compute a new $P_i(\mathbf{A})\mathbf{t}$, we compute $u^T P_i(\mathbf{A})\mathbf{t}$ and then XOR it to the appropriate registers of the sequence. This operation requires a fan-out of depth $\mathcal{O}(\log \ell)$, which accounts for the additional depth $\ell \log \ell$.

Overall there will be $\mathcal{O}(\ell^2)$ CX operations performed. The complexity is dominated by the $\mathsf{Mult}_{\mathbf{A}}$ operations. Once we have constructed the entire sequence, we perform the $\mathsf{Mult}_{\mathbf{A}}$s in reverse to erase the intermediate registers. □

Lemma 5. *Let $\mathbf{A}$ be a matrix of dimension $n-k$. Given an implementation of $\mathsf{Mult}_{\mathbf{A}}$ with G gates, depth D and $\mathcal{O}(n-k)$ space, there exists a reversible circuit to compute $C(\mathbf{A})\mathbf{t}$ on an input polynomial $C(X)$ of degree $\leq \ell$ using $\mathcal{O}(\ell G + ((n-k)+\ell)\ell)$ gates, depth $\mathcal{O}(\ell D + \ell \log(n-k))$ and $\mathcal{O}(n-k+\ell)$ space.*

Proof. The technique is very similar, using the fact that each $\mathbf{A}^i\mathbf{t}$ is a linear combination of the $P_j(\mathbf{A})\mathbf{t}$ with fixed coefficients.

Let us write $M_\ell = (m_{ij})_{0 \leq i,j \leq \ell}$ and $C(X) = \sum_{i=0}^{\ell} c_i X^i$, then:

$$C(\mathbf{A})\mathbf{t} = \sum_{i=0}^{\ell} c_i \mathbf{A}^i \mathbf{t} = \sum_{i=0}^{\ell} \sum_{j=0}^{\ell} m_{ij} c_i P_j(\mathbf{A})\mathbf{t} = \sum_{j=0}^{\ell} \left(\sum_{i=0}^{\ell} m_{ij} c_i \right) P_j(\mathbf{A})\mathbf{t} . \quad (19)$$

We start by computing the vector of all $c'_j := \left(\sum_{i=0}^{\ell} m_{ij} c_i\right)$ for $0 \leq j \leq \ell$, and storing this in $\ell+1$ qubits. Afterwards we compute the sequence of the $P_j(\mathbf{A})\mathbf{t}$, and depending on the stored coefficients, add this to our output register. The additional depth $\ell \log(n-k)$ comes from having to fan-out the current coefficient to control the addition to the output. □

4.2 Reversible Berlekamp-Massey

In order to explain our reversible implementation, we recall the Berlekamp-Massey algorithm [8] in Algorithm 4. We use N do denote the length of the input sequence, which will be $\mathcal{O}(n-k)$ in our case.

Similar to the reversible version of Euclide's algorithm [50], we run a sequence of iterations where each one creates only $\mathcal{O}(1)$ bits of garbage, which can be stored. In our case, there are two such Boolean values: d, and a value v which decides if we enter case 2 or case 3 (leading to a modification of the polynomials, and of L).

First of all, since we focus on the binary case, the coefficient b is always 1 in the algorithm. Second, we notice that we can remove the variable m, by

Algorithm 4 Classical Berlekamp-Massey algorithm

Input: sequence $s_0, \dots, s_{N-1}$ in $\mathbb{F}_q$
Output: retroaction polynomial $C(X) \in \mathbb{F}_q[X]$
$C(X) \leftarrow 1$, $B(X) \leftarrow 1$
$L \leftarrow 0; m \leftarrow 1; b \leftarrow 1$
for all $k = 1 \dots N-1$ **do**
    $d \leftarrow s_k + \sum_{i=1}^{L} c_i s_{k-i}$
    if $d = 0$ **then** ▷ Case 1
        $m \leftarrow m + 1$
    else if $2L \leqslant k$ **then** ▷ Case 2
        $B(X), C(X) \leftarrow C(X), C(X) - \frac{d}{b} X^m B(X)$
        $L \leftarrow k + 1 - L; b \leftarrow d; m \leftarrow 1$
    else ▷ Case 3
        $C(X) \leftarrow C(X) - \frac{d}{b} X^m B(X)$
        $m \leftarrow m + 1$
    end if
end for
Return Reversed$(C(X))$

performing instead the operation $B(X) \leftarrow XB(X)$ each time we would have incremented m. This turns the algorithm into a less efficient version, but more suitable for reversibility.

Finally, we reorder the operations in the loop, as we notice that the shift $B(X) \leftarrow XB(X)$ is performed in all cases, and the operation $C(X) \leftarrow C(X) + B(X)$ is performed in all cases where $d = 1$. We obtain Algorithm 5.

Lemma 6. *Algorithm 5 can be implemented as a quantum circuit using $\mathcal{O}(N^2)$ quantum gates, $\mathcal{O}(N)$ space and depth $\mathcal{O}(N \log N)$.*

Proof. First of all, it is clear that each of the N loop iterations applies reversibly on the registers C, B, L, d_i, v_i

After performing these iterations, we copy the output $C(X)$. Then we compute the reverse of the iterations to erase all intermediate registers. After, we still need to reverse the polynomial $C(X)$. This is done in place using the implementation described in Lemma 12.

In order to simplify the implementation, L is represented in *unary*, i.e., as a list of N bits $(\ell_1, \dots, \ell_N)$ where $\ell_i = 1 \iff L \leq i$. This allows to perform the computation of d_k in $\mathcal{O}(N)$ gates (we perform the sum from $i = 1$ to N but use Toffoli gates with ℓ_i as inputs). The depth is $\mathcal{O}(\log N)$ using a fan-in circuit.

Then, v_k can be computed with $\mathcal{O}(1)$ operations since we can just access $\ell_{\lfloor k/2 \rfloor}$.

The shift of B can be implemented by swaps (which are not counted in the total number of gates, as they simply amount to renumbering the qubits). The two conditional XORs costs $\mathcal{O}(N)$ gates and depth $\mathcal{O}(\log N)$, needing again fan-outs of the control. In order to update the unary representation of L, we only need $\mathcal{O}(N)$ gates, as we will apply X gates on the bits at positions before $k + 1$,

Algorithm 5 Reversible Berlekamp-Massey algorithm for $\mathbb{F}_2$.

Input: sequence $s_0, \dots, s_{N-1}$ in $\mathbb{F}_2$
Output: retroaction polynomial $C(X) \in \mathbb{F}_2[X]$
Storage: register for $C(X)$ (N bits), $B(X)$ (N bits), L (N bits, in unary representation)
Garbage: register for $d_1, \dots, d_N$, $d_0 := 1$, register for $v_0, \dots, v_{N-1}$
$C(X) \leftarrow 1$, $B(X) \leftarrow 1$
$L \leftarrow 0$
for all $k = 0 \dots N-1$ **do**
    $d_k \leftarrow s_k + \sum_{i=1}^{L} c_i s_{k-i}$
    $v_k \leftarrow (2L \leqslant k)$ ▷ Boolean value deciding between case 2 and case 3
    $B(X) \leftarrow XB(X)$
    Conditioned on $d_k = 1$ **do**
        $C(X) \leftarrow C(X) + B(X)$
    EndConditioned
    Conditioned on $d_k v_k = 1$ **do** ▷ Remaining operations of case 2
        $B(X) \leftarrow B(X) + C(X)$
        $L \leftarrow k + 1 - L$ ▷ Can be done in place ($k+1$ is a constant here)
    EndConditioned
end for
Return Reversed$(C(X))$

then swap the entire sub-list (though k varies during the loop, it is a constant of the circuit). The depth is $\mathcal{O}(\log N)$, since this is also controlled and we need to fan-out the control.

Finally, the reversion of C costs $\mathcal{O}(N \log N)$ gates and depth $\mathcal{O}(N)$. We use no more than $\mathcal{O}(N)$ ancillas throughout the circuit. □

4.3 Benchmarks

We denote by $Q = Q_J + 2(n-k) + A$ the total number of qubits used by the $\mathsf{Mult}_{\mathbf{H}}$ circuit, where A is the number of ancilla qubits and Q_J the number of qubits used to represent J. We also denote by $G_{\mathrm{X}}, G_{\mathrm{CX}}$ and G_{CCX} its respective X, CX and CCX gate counts.

Using our implementation of Berlekamp-Massey and Wiedemann's algorithms, we obtain the following counts. We neglect terms of smaller magnitude, except for the qubit count which is exact.

$$\begin{cases} \text{Depth} & = 24D(n-k) + 152(n-k)\log_2(n-k) \\ \text{Qubits} & = Q - A + 7(n-k+1) + \max(A + 3(n-k) + 2, 10(n-k) + 11) \\ \text{CCX Gates} & = 24(n-k)G_{\mathrm{CCX}} + 116(n-k)^2 \\ \text{CX Gates} & = 24(n-k)G_{\mathrm{CX}} + 356(n-k)^2 \\ \text{X Gates} & = 24(n-k)G_{\mathrm{X}} \end{cases} \tag{20}$$

As our implementations of $\mathsf{Mult}_{\mathbf{H}}$ will typically have quadratic gate count and depth at least linear in $(n-k)$, we can observe that this cost quickly dominates

over the rest of the algorithm, though the additional terms are not negligible. The constant factors are also quite large, owing to the number of polynomial sequences evaluated during Wiedemann's algorithm and their size (twice $n-k$ to ensure success in the Berlekamp-Massey algorithm).

5 Implementing the Multiplication Circuit

In this section, we implement the multiplication circuit (with an implicit matrix). We propose two main approaches, using different representations of the choice of sub-matrix, i.e., different definitions of the set $\mathcal{J}$.

The first one (Section 5.1) is the approach chosen in previous works [47], where $\mathcal{J}$ is the set of n-bit strings of Hamming weight $n-k$. In that case, Init is a unitary creating a so-called *Dicke state*, whose implementation can be borrowed from these previous works. Using this representation, we are able to decrease the space complexity of Wiedemann's algorithm (hence, the entire quantum Prange) to $\mathcal{O}(n)$.

The second one (Section 5.2) is based on permutations and sorting. In this approach, $\mathcal{J}$ maps to a set of permutations of $\{0,\dots,n-1\}$. Each permutation π of $\{0,\dots,n-1\}$ naturally specifies a subset $I=\{\pi(0),\dots,\pi(n-k-1)\}\in S_{n,k}$. To the best of our knowledge, this idea has appeared in [46] but was not completely exploited. Our result shows a remarkable trade-off between qubit and gate count, where the qubit count increases to $\mathcal{O}\big(n\log^2 n\big)$, but remains comparable in practice to the space-efficient approach; while the total gate count remains at $\mathcal{O}\big(n^3\big)$, the constant factor is reduced, and the CCX gate count becomes asymptotically lower.

Using this second approach, further optimizations are possible (Section 5.3), although they do not perform well for practical parameters at the moment.

5.1 Space-Optimized Circuits

In this subsection, $\mathcal{J}$ is the set of n-bit strings of Hamming weight $n-k$, which is identified with $S_{n,k}$.

Lemma 7. *There exists a reversible circuit implementing the* $\mathsf{Mult}_{\mathbf{H}}$ *operation:*

$$|J\rangle\,|\mathbf{x}\rangle\,|\mathbf{y}\rangle \xmapsto{\mathsf{Mult}_{\mathbf{H}}} |J\rangle\,|\mathbf{y}+\mathbf{H}_J\mathbf{x}\rangle\,|\mathbf{x}\rangle \tag{21}$$

which uses $\mathcal{O}(n)$ *space,* $\mathcal{O}(n(n-k))$ *gates and depth* $\mathcal{O}(n\log(n-k))$.

Proof. The operation that we implement is basically the computation of $\mathbf{H}_J\mathbf{x}$, except that we will directly XOR the result to $\mathbf{y}$.

Let $\mathbf{x}:=(x_0,\dots,x_{n-k-1})$. Furthermore, let $\mathbf{a}=(a_0,a_1,\dots,a_{n-k-1})$ be a vector of integers where a_0 is the position of the first "1" in J, a_1 the second one, etc. We note the coefficients of $\mathbf{H}$ as (h_{ij}) and $\mathbf{H}_J=(h'_{ij})$, then by definition of a_j:

$$\forall 0\le i\le n-k-1, \forall 0\le j\le n-k-1, h'_{ij}=h_{ia_j}=\bigoplus_{k=0}^{n-1}\delta_{a_jk}h_{ik}\ . \tag{22}$$

Thus, we can express $\mathbf{H}_J\mathbf{x}$ as follows:

$$\forall 0 \leq i \leq n-k-1, (\mathbf{H}_J\mathbf{x})_i = \bigoplus_{j=0}^{n-k-1} h'_{ij} x_j = \bigoplus_{j=0}^{n-k-1} \bigoplus_{\ell=0}^{n-1} \delta_{a_j \ell} h_{i\ell} x_j$$
$$= \bigoplus_{\ell=0}^{n-1} \left(\bigoplus_{j=0}^{n-k-1} \delta_{a_j \ell} x_j \right) h_{i\ell} \ . \quad (23)$$

Our strategy is to compute the vector $\mathbf{v} := \left(\bigoplus_{j=0}^{n-k-1} \delta_{a_j \ell} x_j\right)_{0 \leq \ell \leq n-1}$. This vector simply places the coordinates of x at the positions marked by J, keeping their order. As an example, if we have $J = (0, 1, 0, 0, 1, 1, \ldots)$, then $\mathbf{v}$ will start with $(0, x_0, 0, 0, x_1, x_2, \ldots)$.

In order to do so, we maintain a unary counter $\mathbf{e}$, implemented as a register with $n-k$ bits, which remains of weight 1, and represents the number c such that $\mathbf{e}_c = 1$, i.e., such that the bit of weight c in $\mathbf{e}$ equals 1.

For $\ell = 0$ to $n-1$, we compute: $v_\ell = j_\ell \mathbf{e} \cdot \mathbf{x}$ where j_ℓ is the ℓth-bit in the register J. Indeed, the dot-product $\mathbf{e} \cdot \mathbf{x}$ selects a new coordinate in $\mathbf{x}$ each time the counter is updated. Then, we perform a shift of $\mathbf{e}$, controlled on j_ℓ, to update the counter c as $c \leftarrow c + j_\ell$. These operations require $\mathcal{O}(n-k)$ CCX gates and $\mathcal{O}(\log(n-k))$ depth (due to the use of fan-in and fan-out circuits).

Once we have computed v_ℓ, we use another fan-out and update the output $\mathbf{H}_J\mathbf{x}$. Indeed, from Equation 23 we have:

$$\forall 0 \leq i \leq n-k-1, (\mathbf{H}_J x)_i = \bigoplus_{\ell=0}^{n-1} v_\ell h_{i\ell} \ . \quad (24)$$

So we simply need to XOR v_ℓ at the right positions. This costs $\mathcal{O}(n-k)$ CXs. We then uncompute the fan-out, erase v_ℓ and go to the next iteration. Since there are n iterations, the overall gate count and depth are respectively $\mathcal{O}(n(n-k))$ and $\mathcal{O}(n \log(n-k))$. □

Cost Formulas. We computed asymptotic formulas for this space-optimized $\mathsf{Mult}_{\mathbf{H}}$ circuit (left), and combined them with Equation 20 to obtain the cost of the inversion circuit (right):

$$\begin{cases} \text{Depth} & = 4n \log_2(n-k) \\ \text{Qubits} & = n + 6(n-k) + 2 \\ \text{CCX Gates} & = 5n(n-k) \\ \text{CX Gates} & = 9n(n-k) \\ \text{X Gates} & = 2 \end{cases} \qquad \begin{cases} \text{Depth} & = 96n(n-k) \log_2(n-k) \\ \text{Qubits} & = n + 19(n-k) + 18 \\ \text{CCX Gates} & = 120n(n-k)^2 \\ \text{CX Gates} & = 216n(n-k)^2 \\ \text{X Gates} & = \mathcal{O}(n-k) \end{cases} \quad (25)$$

The multiplication of constants between the $\mathsf{Mult}_{\mathbf{H}}$ circuit and the inversion circuit creates even larger constants, which are far from negligible for actual parameters.

5.2 Toffoli-Optimized Circuits

In this second approach, the set $\mathcal{J}$ is defined by means of a *sorting network*. Note that [47] used similar tools to permute the columns of the matrix $\mathbf{H}$ within the QAA iteration; our reasoning is different here since we directly implement $\mathsf{Mult}_{\mathbf{H}}$.

A sorting network with n entries is defined as a sequence of *comparators* and *switches*, which respectively compare a pair of entries at fixed positions, and swap them depending on the result of the comparison. While sorting networks with $\mathcal{O}(n \log n)$ comparators exist [2], one of the most efficient in practice is Batcher's odd-even mergesort [5], which has depth $\lceil \log_2 n \rceil (\lceil \log_2 n \rceil + 1)/2$ and contains $\frac{n}{4} \lceil \log_2 n \rceil (\lceil \log_2 n \rceil - 1) + n - 1$ comparators. This is the one we use here.

Let $(A_0, \ldots, A_{n-1})$ be an n-tuple of integers. Let us define the mapping $\mathcal{N}$ from $(A_0, \ldots, A_{n-1})$ to bit-strings of length $\frac{n}{4} \lceil \log_2 n \rceil (\lceil \log_2 n \rceil - 1) + n - 1 = \mathcal{O}\big(n \log^2 n\big)$ which gives the results of all comparisons in the sorting network, where the comparators are taken in a fixed, arbitrary order. While the sorting network itself is not reversible, storing $\mathcal{N}(A_0, \ldots, A_{n-1})$ is sufficient to make it reversible. This increases the space usage of Batcher's network to $\mathcal{O}\big(n \log^2 n\big)$.

Definition of Init. Equipped with the mapping $\mathcal{N}$ above, we now define $\mathcal{J}$ as:

$$\mathcal{J} = \Big\{ \Big((A_0, \ldots, A_{n-1}), \mathcal{N}(A_0, \ldots, A_{n-1}) \Big), A_0, \ldots, A_{n-1} \in [0; n^3 - 1] \Big\} \quad . \quad (26)$$

That is, we take an n-tuple of integers $(A_0, \ldots, A_{n-1})$ between 0 and n^3, append the result of all comparisons in the network, and identify this as a bit-string.

The unitary Init, which creates the uniform superposition over $\mathcal{J}$, essentially consists in taking uniform superposition of such integers (which is efficient) and computing a reversible sorting network. In particular, the bit-string $\mathcal{N}(A_0, \ldots, A_{n-1})$ is computed only once, at this step, and used later in the multiplication circuit without the need to recompute it. As a comparison of integers can be performed without ancillas using a modified CDKM addition circuit [22], Init uses almost no ancillas.

Besides removing the need for Dicke states, this definition will make the multiplication circuit less costly, as we show later.

Mapping to a Subset of Columns. We explain here how an element $J \in \mathcal{J}$ defines a subset $S_{n,k}$, and why all subsets have the same probability to appear. This mapping is especially important for the definition of $\mathsf{Mult}_{\mathbf{H}}$.

First of all, an element $J \in \mathcal{J}$ defines a permutation π_J of $\{0, \ldots, n-1\}$, which is the permutation such that sorting $A_0, \ldots, A_{n-1}$ puts the integer A_i in position $\pi_J(i)$. This permutation can easily be implemented by a *switching network*. This network has the same structure as the sorting network that defines

$\mathcal{N}$, but it is made only of controlled swaps (the switches), which are controlled by the bits of $\mathcal{N}(A_0, \ldots, A_{n-1})$.

It is well-known that, if we sort n *distinct* entries chosen uniformly at random, the permutation π_J is also uniformly random. By choosing entries with sufficiently many bits, they will all be distinct with large probability.

Lemma 8. *Let $(A_0, \ldots, A_{n-1})$ be drawn uniformly at random in $[0; n^3-1]$, then they are all distinct with probability at least $1 - \frac{1}{2n}$.*

Proof. We simply lower bound the probability of all A_i to be distinct, as:

$$\left(1 - \frac{1}{n^3}\right)\left(1 - \frac{2}{n^3}\right)\cdots\left(1 - \frac{n-1}{n^3}\right) \geq 1 - \sum_{i=1}^{n-1} \frac{i}{n^3} \geq 1 - \frac{1}{2n} \quad . \quad \square$$

In the case where the entries are not distinct, we do not know if the algorithm will succeed. Luckily, our implementation of Wiedemann's inversion ensures that there are no false positives, so we can still use QAA (Theorem 3). Indeed, we know that the oracle f returns 1 for the tuples $(A_0, \ldots, A_{n-1})$ for which all the numbers are distinct, and the corresponding permutation returns a solution, so the probability of success of the amplified algorithm is at least $\left(1 - \frac{1}{2n}\right) 2^{-3.50 \frac{\binom{n-k}{w}}{\binom{n}{w}}}$.

Finally, the permutation π_J defines a subset of columns from $S_{n,k}$ as follows: the positions of the columns are $\pi_J(0), \pi_J(1), \ldots, \pi_J(n-k-1)$. As π_J is a uniformly random permutation (when selecting J at random from $\mathcal{J}$), the subset $\{\pi_J(0), \pi_J(1), \ldots, \pi_J(n-k-1)\}$ is also a uniformly random element of $S_{n,k}$.

Definition of the Multiplication Circuit. We make a small tweak to the definition of the sub-matrix $\mathbf{H}_J$. Since we defined a permutation of columns, it makes sense to define $\mathbf{H}_J$ as:

$$(\mathbf{H}_J)_{ij} := (h_{i\pi_J(j)}) \tag{27}$$

This definition is slightly different from the one of Section 5.1, where the columns were put in a fixed order. Here, the columns of $\mathbf{H}_J$ will also be permuted. This has no incidence on the rest of the algorithm.

We can now implement our circuit for $\mathsf{Mult}_{\mathbf{H}}$, which takes as input an element of $\mathcal{J}$. In fact, this circuit does not need the integers $(A_0, \ldots, A_{n-1})$, which we are keeping along only for the sake of reversibility. It only relies on the bit-string $\mathcal{N}(A_0, \ldots, A_{n-1})$ which defines the permutation π_J.

Lemma 9. *There exists a reversible circuit implementing the $\mathsf{Mult}_{\mathbf{H}}$ operation:*

$$|J\rangle\,|\mathbf{x}\rangle\,|\mathbf{y}\rangle \xmapsto{\mathsf{Mult}_{\mathbf{H}}} |J\rangle\,|\mathbf{y} + \mathbf{H}_J\mathbf{x}\rangle\,|\mathbf{x}\rangle \tag{28}$$

using $\mathcal{O}\left(n \log^2 n\right)$ space, $\mathcal{O}\left(n \log^2 n\right)$ CCX gates, $\mathcal{O}\left(n^2\right)$ CX gates and depth $n + o(n)$.

Proof. The idea of the circuit is very similar to Lemma 7. First, we compute the vector $\mathbf{v}$ that places the input bits $\mathbf{x}$ at appropriate positions, i.e., bit x_i in position $\pi_J(i)$. Then, we compute the fixed matrix-vector product $\mathbf{Hv}$.

The first step is done using the switching network, i.e., a series of $\mathcal{O}\big(n \log^2 n\big)$ controlled swaps with depth $\mathcal{O}\big(\log^2 n\big)$.

The second step can be done in depth n and $\mathcal{O}\big(n^2\big)$ CX gates as shown in Lemma 13. □

Interestingly, the dominating operation becomes the product of $\mathbf{v}$ by the constant matrix $\mathbf{H}$. This is a linear quantum circuit, which can be implemented with only CX gates. The depth is also asymptotically optimal. This appears clearly on our asymptotic cost formulas for this alternative function:

$$\begin{cases} \text{Depth} & = n \\ \text{Qubits} & = n + (n-k) + \frac{n}{4}\lceil \log_2 n \rceil \left(\lceil \log_2 n \rceil - 1\right) + n - 1 + 3n \log_2 n \\ \text{CCX Gates} & = \frac{1}{2} n (\log_2 n)^2 \\ \text{CX Gates} & = n(n-k) \\ \text{X Gates} & = \mathcal{O}(1) \end{cases} \tag{29}$$

Having much lower constants than Equation 25, these counts yield much more favorable results when we plug them in Equation 20:

$$\begin{cases} \text{Depth} & = 24n(n-k) \\ \text{Qubits} & = \frac{n}{4}\lceil \log_2 n \rceil \left(\lceil \log_2 n \rceil - 1\right) + n + 19(n-k) + 17 + 3n \log_2 n \\ \text{CCX Gates} & = 12n(n-k)(\log_2 n)^2 + 116(n-k)^2 \\ \text{CX Gates} & = 24n(n-k)^2 + 356(n-k)^2 \\ \text{X Gates} & = \mathcal{O}(n-k) \end{cases} \tag{30}$$

In both these formulas, the term $3n \log_2 n$ in the qubit count comes from the initial tuple of integers $(A_0, \ldots, A_{n-1})$. They actually do not intervene in the definition of the circuits, but we need to keep them along in order to be able to invert the Init circuit. This term is asymptotically negligible, but not entirely when $n \simeq 10^3$.

5.3 Gate-Optimized Multiplication Circuit for Circulant Matrices

In the case of BIKE [3] and HQC [1], one has $n = 2k$ and the parity-check matrix $\mathbf{H}$ is made of two $k \times k$ circulant blocks. Therefore, we can replace the multiplication by $\mathbf{H}$ by a more efficient circuit using Karatsuba multiplication of polynomials (detailed in Corollary 1 in Section A). While our benchmarks show a noticeable improvement in total gate count, the downside is an increase in depth, since the Karatsuba circuit that we use, based on Gidney [29], has asymptotically worse depth.

Asymptotically, binary polynomial multiplication can be performed in $\widetilde{\mathcal{O}}(n)$ binary operations, for example using Cantor's algorithm [19] in $\mathcal{O}\big(n(\log n)^{1.585}\big)$. This means that there exists a circuit for multiplication by a circulant matrix using $\mathcal{O}\big(n(\log n)^{1.585}\big)$ gates and qubits, and consequently:

Theorem 4. *If the parity-check matrix is block-circulant, there exists a quantum algorithm solving SD for random codes using* $\mathcal{O}\left(n^2(\log n)^2 \times \sqrt{\frac{\binom{n}{k}}{\binom{n-t}{k}}}\right)$ *gates and* $\mathcal{O}\big(n(\log n)^2\big)$ *qubits.*

This decrease of the gate count is specific to our "sorting-based" approach, using the fact that $\mathbf{H}$ is structured and that Wiedemann's algorithm can make use of this. To the best of our knowledge, this is the first asymptotic improvement over the $\mathcal{O}\big(n^3\big)$ linear algebra factor in quantum ISD to date.

Unfortunately, while efficient classical software exists [18], corresponding quantum circuits for circulant matrix-vector multiplication have not been studied as much. In particular, the constant factors, depth and qubit counts of this method remain unknown.

6 Evaluation of Costs for Code-Based Cryptosystems

In this section we give resource estimates for the three inversion circuits detailed in Section 5, and compare them.

6.1 Comparison of Circuits

We computed the number of gates, qubits and depth of our circuits for parameters of the three round 4 candidates for post-quantum key-exchange based on codes at the NIST post-quantum standardization: Classic McEliece [13], BIKE [3] and HQC [1]. We compare them with the counts of [47] in Table 1.

Even if we disregard the use of memory, it is difficult to compare our results with the advanced quantum ISD algorithms that could apply here [35,37,36], since they considered only asymptotic complexities and neglected polynomial factors. However, it is likely that these algorithms could benefit from improved linear algebra circuits.

We note that, while we compare here with [47], Bonnetain and Jaques also designed a quantum circuit for binary Gaussian elimination for a matrix of dimension $(n-k) \times (n-k)$ with depth $\mathcal{O}((n-k)\log(n-k))$ [15]. This is better than the depth $\mathcal{O}\big((n-k)^2\big)$ reported in [47], so we believe their counts could be immediately improved by using the circuit of [15] as a replacement. Nevertheless, our main focus in Table 1 is on the number of qubits.

Let us consider the Classic McEliece parameters for NIST security level 1, which are at least as secure as AES-128 against Grover's exhaustive key search ("McEliece L1" in Table 1). Using the space-optimized circuit, the total number of qubits required for quantum Prange is $18 + n + 19(n-k) = 18\,098$, instead of $2^{22} \simeq 4\,194\,304$ reported in [47]. Previously one would have needed at least $(n-k)^2 = 589\,824$ qubits at best to store the matrix being inverted using Gaussian elimination. Our improvement brings the number of logical qubits to the same order as the one required in factoring large instances of RSA [30] via Shor's algorithm [52].

Table 1. Quantum resource estimates for the QAA iteration. Counts are given in $\log_2$ and rounded. The number of CCX gates is not given in [47], but due to the structure of the Gaussian elimination circuit, it is of the same order as the total number of gates.

Implementation	Scheme	n	k	Counts (in $\log_2$) CCX	Total gates	Depth	Qubits	DW
[47]	BIKE L1	24646	12323		43	28	29	57
	BIKE L3	49318	24659		46	31	31	62
	BIKE L5	81946	40973		48	32	33	65
	HQC L1	35338	17669		45	30	30	60
	HQC L3	71702	35851		47	32	32	64
	HQC L5	115274	57637		50	34	34	68
	McEliece L1	3488	2720		30	20	22	42
	McEliece L3	4608	3360		32	22	23	45
	McEliece L5-1	6688	5024		34	23	24	47
	McEliece L5-2	6960	5413		33	23	24	47
	McEliece L5-3	8192	6528		34	23	24	47
Space-optimized Section 5.1	BIKE L1	24646	12323	48.7	50.2	38.9	18.0	56.9
	BIKE L3	49318	24659	51.7	53.2	41.0	19.0	60.0
	BIKE L5	81946	40973	53.9	55.4	42.5	19.7	62.2
	HQC L1	35338	17669	50.2	51.7	40.0	18.5	58.5
	HQC L3	71702	35851	53.3	54.8	42.1	19.5	61.7
	HQC L5	115274	57637	55.4	56.8	43.5	20.2	63.7
	McEliece L1	3488	2720	37.8	39.3	31.7	14.1	45.8
	McEliece L3	4608	3360	39.6	41.1	32.9	14.8	47.7
	McEliece L5-1	6688	5024	41.0	42.5	33.9	15.2	49.1
	McEliece L5-2	6960	5413	40.9	42.3	33.8	15.2	49.0
	McEliece L5-3	8192	6528	41.3	42.8	34.1	15.3	49.4
Toffoli-optimized Section 5.2	BIKE L1	24646	12323	39.5	46.4	32.8	21.8	54.6
	BIKE L3	49318	24659	41.7	49.4	34.8	22.9	57.7
	BIKE L5	81946	40973	43.4	51.6	36.2	23.8	60.0
	HQC L1	35338	17669	40.8	47.9	33.8	22.4	56.3
	HQC L3	71702	35851	43.0	51.0	35.9	23.6	59.4
	HQC L5	115274	57637	44.3	53.0	37.2	24.3	61.5
	McEliece L1	3488	2720	32.0	35.9	26.2	18.5	44.7
	McEliece L3	4608	3360	33.4	37.6	27.2	19.1	46.3
	McEliece L5-1	6688	5024	34.3	38.9	28.1	19.6	47.8
	McEliece L5-2	6960	5413	34.3	38.8	28.1	19.7	47.8
	McEliece L5-3	8192	6528	34.6	39.2	28.4	19.9	48.3
Karatsuba Section 5.3	BIKE L1	24646	12323	39.5	44.3	40.3	21.8	62.1
	BIKE L3	49318	24659	41.7	46.8	42.9	23.0	65.8
	BIKE L5	81946	40973	43.4	49.1	45.2	23.8	69.0
	HQC L1	35338	17669	40.8	46.3	42.4	22.4	64.8
	HQC L3	71702	35851	43.0	48.9	45.0	23.6	68.6
	HQC L5	115274	57637	44.3	49.6	45.7	24.3	70.0

However, this optimization of space comes at the expense of gate count and depth. Indeed, both increase a thousandfold, mostly due to the large constant factors appearing in Equation 25. Overall, the product between depth and width of the circuit (so-called "DW" metric) increases slightly.

The sorting-based approach has a much better trade-off. On the same example, it will use 258 769 qubits, among which 115 104 are used to store the state of switches, and 125 568 to store the initial numbers which are sorted. This increase in the space complexity comes entirely from our representation of the column choice, which could likely be compacted. On this example, the total gate count and depth are significantly reduced but remain a factor 2^6 above those of [47]. The difference is more favorable for larger code lengths as the Toffoli count is asymptotically smaller.

With the same amount of qubits, the use of Karatsuba-based multiplication of polynomials for the matrix-vector product reduces the gate count asymptotically. The difference is already noticeable for the BIKE and HQC parameters. However, our implementation is not optimized in depth. As a consequence the DW product increases significantly.

6.2 Discussion

Our work does not threaten the security of the NIST code-based candidates Classic McEliece, BIKE and HQC. In fact, it does not overall improve the circuit depth with respect to [47] and [15], and the gains in DW product that we observed with respect to [47] come mostly from the reduction in qubits. Besides, we lose the gain of DOOM that is exploitable with Gaussian elimination in the case of BIKE and HQC, as mentioned in Section 3.2. However, DOOM reduces the number of iterations by a factor $\sqrt{n}$, while our method reduces the Toffoli gate count (and the total gate count for block-circulant matrices) by a factor of order $\frac{n}{\log^2 n}$, which is asymptotically better.

While our space-optimized circuit reaches quite competitive qubit counts, we have observed that the Toffoli-optimized approach offers a better trade-off in practice, and can be combined with an improved matrix multiplication circuit for block circulant matrices. There are several ways in which this approach can be further improved.

First of all, the bottleneck of the cost in Toffoli (CCX) gates is the switching network that is used in $\mathsf{Mult}_{\mathbf{H}}$. Right now, this network contains $\mathcal{O}\left(n \log^2 n\right)$ controlled swaps. However, it is known that given a permutation π of $\{0, \ldots, n-1\}$, one can design a network with only $\mathcal{O}(n \log n)$ swaps that implements π. Such an algorithm is described in detail in [12], but the difficulty would be to implement it as an efficient quantum circuit. We would use this circuit once in the QAA iteration and store the network using $\mathcal{O}(n \log n)$ qubits. The CCX gate of the $\mathsf{Mult}_{\mathbf{H}}$ operation would further decrease to $\mathcal{O}(n \log n)$.

The bottleneck in the space complexity is the integers $(A_0, \ldots, A_{n-1})$ which we use as intermediates to sample a random permutation, and the state of the comparators which we use to represent it. Other ways to generate a random permutation (e.g., the Fisher–Yates shuffle) did not seem competitive. However,

our approach right now is quite conservative, as we ensured that the permutation was sampled uniformly at random. This requirement can be relaxed: we only want to sample from a family of permutations that distribute well the subset of $n-k$ columns to be selected, so that the probability of finding a solution remains high. It is known that switching networks with $\mathcal{O}(n \log n)$ and depth $\mathcal{O}\big(\log^2 n\big)$ with good mixing properties can be constructed [23]. We believe that such a construction could be used to reduce both the CCX gate count and number of qubits, but leave this as future work.

7 Conclusion

In this paper, we achieved new trade-offs in the linear algebra circuit required in the quantum Prange's algorithm. In particular, we can bring the number of qubits down to $\mathcal{O}(n)$, at a level similar to what Shor's algorithm requires for large RSA instances. The core idea is to use Wiedemann's matrix inversion algorithm, where the matrix to invert is only implicitly represented. Our main contribution is a complete reversible and space-efficient implementation of this algorithm with detailed gate counts.

While our new approach removes the limitation of the number of qubits, we still expect quantum ISD to remain unrealizable for code-based cryptosystems, even for large-scale quantum computers, due to its large circuit depth and gate count requirements.

Nevertheless, our result greatly improves the known time-memory trade-offs [27], and switches the focus towards the time complexity. In this context, we also showed that Wiedemann inversion, combined with an appropriate representation of column permutations in Prange's algorithm, improves the Toffoli (CCX) gate count with respect to Gaussian elimination. It can also improve the overall gate count in the case of circulant matrices. Our estimations shows that these improvements are observable for actual parameters of code-based cryptosystems, but further dedicated circuit optimizations could significantly enhance these results.

Finally, although this paper focused on the quantum Prange algorithm, our implementation of Wiedemann's algorithm is of independent interest, as there are other quantum algorithms that need to inverse a sparse or implicit matrix, for example solving multivariate polynomial equation systems [28,14]. Our circuit could be used to reduce the memory complexity, and perhaps estimate more precisely the time complexity of such methods.

Acknowledgments. We would like to thank the anonymous reviewers of ASIACRYPT 2024 for helpful remarks. This work has been supported by the French Agence Nationale de la Recherche through the CROWD project under Contract ANR-CE 48 2022, and through the France 2030 program under grant agreement No. ANR-22-PETQ-0008 PQ-TLS.

References

1. Aguilar Melchor, C., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Persichetti, E., , Zémor, G., Bos, J., Dion, A., Lacan, J., Robert, J.M., Véron, P.: Hamming quasi-cyclic (HQC). Submission to the NIST PQC process, Round 4 (2022), https://pqc-hqc.org/
2. Ajtai, M., Komlós, J., Szemerédi, E.: An O(n log n) sorting network. In: STOC. pp. 1–9. ACM (1983). https://doi.org/10.1145/800061.808726
3. Aragon, N., Barreto, P., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Gueron, S., Güneysu, T., Aguilar Melchor, C., Misoczki, R., Persichetti, E., Sendrier, N., Tillich, J.P., Zémor, G., Vasseur, V., Ghosh, S., Richter-Brokmann, J.: BIKE: bit flipping key encapsulation. Submission to the NIST PQC process, Round 4 (2022), https://bikesuite.org/
4. Bärtschi, A., Eidenbenz, S.J.: Short-depth circuits for Dicke state preparation. In: QCE. pp. 87–96. IEEE (2022). https://doi.org/10.1109/QCE53715.2022.00027
5. Batcher, K.E.: Sorting networks and their applications. In: AFIPS Spring Joint Computing Conference. AFIPS Conference Proceedings, vol. 32, pp. 307–314. Thomson Book Company, Washington D.C. (1968). https://doi.org/10.1145/1468075.1468121
6. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 7237, pp. 520–536. Springer (2012). https://doi.org/10.1007/978-3-642-29011-4_31
7. Bennett, C.H.: Time/space trade-offs for reversible computation. SIAM J. Comput. **18**(4), 766–776 (1989)
8. Berlekamp, E.R.: Algebraic coding theory. McGraw-Hill series in systems science, McGraw-Hill (1968), https://www.worldcat.org/oclc/00256659
9. Berlekamp, E.R., McEliece, R.J., van Tilborg, H.C.A.: On the inherent intractability of certain coding problems (corresp.). IEEE Trans. Inf. Theory **24**(3), 384–386 (1978). https://doi.org/10.1109/TIT.1978.1055873
10. Bernstein, D.J.: Batch binary edwards. In: CRYPTO. Lecture Notes in Computer Science, vol. 5677, pp. 317–336. Springer (2009). https://doi.org/10.1007/978-3-642-03356-8_19
11. Bernstein, D.J.: Grover vs. mceliece. In: PQCrypto. Lecture Notes in Computer Science, vol. 6061, pp. 73–80. Springer (2010). https://doi.org/10.1007/978-3-642-12929-2_6
12. Bernstein, D.J.: Verified fast formulas for control bits for permutation networks. IACR Cryptol. ePrint Arch. p. 1493 (2020), https://eprint.iacr.org/2020/1493
13. Bernstein, D.J., Chou, T., Cid, C., Gilcher, J., Lange, T., Maram, V., von Maurich, I., Misoczki, R., Niederhagen, R., Persichetti, E., Peters, C., Sendrier, N., Szefer, J., Tjhai, C.J., Tomlinson, M., Wang, W.: Classic McEliece: conservative code-based cryptography. Submission to the NIST PQC process, Round 4 (2022), https://classic.mceliece.org
14. Bernstein, D.J., Yang, B.: Asymptotically faster quantum algorithms to solve multivariate quadratic equations. In: PQCrypto. Lecture Notes in Computer Science, vol. 10786, pp. 487–506. Springer (2018). https://doi.org/10.1007/978-3-319-79063-3_23
15. Bonnetain, X., Jaques, S.: Quantum period finding against symmetric primitives in practice. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2022**(1), 1–27 (2022). https://doi.org/10.46586/TCHES.V2022.I1.1-27

16. Both, L., May, A.: Decoding linear codes with high error rate and its impact for LPN security. In: PQCrypto. Lecture Notes in Computer Science, vol. 10786, pp. 25–46. Springer (2018). https://doi.org/10.1007/978-3-319-79063-3_2
17. Brassard, G., Høyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. Contemporary Mathematics **305**, 53–74 (2002). https://doi.org/10.1090/conm/305/05215
18. Brent, R.P., Gaudry, P., Thomé, E., Zimmermann, P.: Faster multiplication in gf(2)[x]. In: ANTS. Lecture Notes in Computer Science, vol. 5011, pp. 153–166. Springer (2008). https://doi.org/10.1007/978-3-540-79456-1_10
19. Cantor, D.G.: On arithmetical algorithms over finite fields. J. Comb. Theory, Ser. A **50**(2), 285–300 (1989). https://doi.org/10.1016/0097-3165(89)90020-4
20. Chailloux, A., Debris-Alazard, T., Etinski, S.: Classical and quantum algorithms for generic syndrome decoding problems and applications to the lee metric. In: PQCrypto. Lecture Notes in Computer Science, vol. 12841, pp. 44–62. Springer (2021). https://doi.org/10.1007/978-3-030-81293-5_3
21. Cooper, C.: On the distribution of rank of a random matrix over a finite field. Random Struct. Algorithms **17**(3-4), 197–212 (2000)
22. Cuccaro, S.A., Draper, T.G., Kutin, S.A., Moulton, D.P.: A new quantum ripple-carry addition circuit (2004)
23. Czumaj, A.: Random permutations using switching networks. In: STOC. pp. 703–712. ACM (2015). https://doi.org/10.1145/2746539.2746629
24. Dornstetter, J.: On the equivalence between Berlekamp's and Euclid's algorithms (corresp.). IEEE transactions on information theory **33**(3), 428–431 (1987)
25. Ducas, L., Esser, A., Etinski, S., Kirshanova, E.: Asymptotics and improvements of sieving for codes. In: EUROCRYPT (6). Lecture Notes in Computer Science, vol. 14656, pp. 151–180. Springer (2024). https://doi.org/10.1007/978-3-031-58754-2_6
26. Esser, A., Ramos-Calderer, S., Bellini, E., Latorre, J.I., Manzano, M.: An optimized quantum implementation of ISD on scalable quantum resources. IACR Cryptol. ePrint Arch. p. 1608 (2021), https://eprint.iacr.org/2021/1608
27. Esser, A., Ramos-Calderer, S., Bellini, E., Latorre, J.I., Manzano, M.: Hybrid decoding - classical-quantum trade-offs for information set decoding. In: PQCrypto. Lecture Notes in Computer Science, vol. 13512, pp. 3–23. Springer (2022). https://doi.org/10.1007/978-3-031-17234-2_1
28. Faugère, J., Horan, K., Kahrobaei, D., Kaplan, M., Kashefi, E., Perret, L.: Fast quantum algorithm for solving multivariate quadratic equations. CoRR **abs/1712.07211** (2017), http://arxiv.org/abs/1712.07211
29. Gidney, C.: Asymptotically efficient quantum karatsuba multiplication. arXiv preprint arXiv:1904.07356 (2019)
30. Gidney, C., Ekerå, M.: How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. Quantum **5**, 433 (2021). https://doi.org/10.22331/Q-2021-04-15-433, https://doi.org/10.22331/q-2021-04-15-433
31. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: STOC. pp. 212–219. ACM (1996). https://doi.org/10.1145/237814.237866
32. Guo, Q., Johansson, T., Nguyen, V.: A new sieving-style information-set decoding algorithm. IACR Cryptol. ePrint Arch. p. 247 (2023), https://eprint.iacr.org/2023/247
33. van Hoof, I.: Space-efficient quantum multiplication of polynomials for binary finite fields with sub-quadratic toffoli gate count. IACR Cryptol. ePrint Arch. p. 1170 (2019), https://eprint.iacr.org/2019/1170

34. Jaques, S., Naehrig, M., Roetteler, M., Virdia, F.: Implementing grover oracles for quantum key search on AES and LowMC. In: EUROCRYPT (2). Lecture Notes in Computer Science, vol. 12106, pp. 280–310. Springer (2020). https://doi.org/10.1007/978-3-030-45724-2_10
35. Kachigar, G., Tillich, J.: Quantum information set decoding algorithms. In: PQCrypto. Lecture Notes in Computer Science, vol. 10346, pp. 69–89. Springer (2017). https://doi.org/10.1007/978-3-319-59879-6_5
36. Kimura, N., Takayasu, A., Takagi, T.: Memory-efficient quantum information set decoding algorithm. In: ACISP. Lecture Notes in Computer Science, vol. 13915, pp. 452–468. Springer (2023). https://doi.org/10.1007/978-3-031-35486-1_20
37. Kirshanova, E.: Improved quantum information set decoding. In: PQCrypto. Lecture Notes in Computer Science, vol. 10786, pp. 507–527. Springer (2018). https://doi.org/10.1007/978-3-319-79063-3_24
38. Lee, P.J., Brickell, E.F.: An observation on the security of McEliece's public-key cryptosystem. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 330, pp. 275–280. Springer (1988). https://doi.org/10.1007/3-540-45961-8_25
39. Massey, J.L.: Shift-register synthesis and BCH decoding. IEEE Trans. Inf. Theory **15**(1), 122–127 (1969). https://doi.org/10.1109/TIT.1969.1054260
40. May, A., Meurer, A., Thomae, E.: Decoding random linear codes in $\mathcal{O}(2^{0.054n})$. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 7073, pp. 107–124. Springer (2011). https://doi.org/10.1007/978-3-642-25385-0_6
41. May, A., Ozerov, I.: On computing nearest neighbors with applications to decoding of binary linear codes. In: EUROCRYPT (1). Lecture Notes in Computer Science, vol. 9056, pp. 203–228. Springer (2015). https://doi.org/10.1007/978-3-662-46800-5_9
42. Nielsen, M.A., Chuang, I.: Quantum computation and quantum information (2002)
43. NIST: Submission requirements and evaluation criteria for the post-quantum cryptography standardization process (2016), https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf
44. NIST: Post-quantum cryptography: Digital signature schemes - round 1 additional signatures (2023), https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures
45. NIST: Round 4 standardisation results for the post-quantum cryptography standardization process (2024), https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions
46. Perriello, S.: Design and development of a quantum circuit to solve the information set decoding problem (2017)
47. Perriello, S., Barenghi, A., Pelosi, G.: Improving the efficiency of quantum circuits for information set decoding. ACM Transactions on Quantum Computing **4**(4), 1–40 (2023)
48. Prange, E.: The use of information sets in decoding cyclic codes. IRE Trans. Inf. Theory **8**(5), 5–9 (1962). https://doi.org/10.1109/TIT.1962.1057777
49. Qiskit contributors: Qiskit: An open-source framework for quantum computing (2023). https://doi.org/10.5281/zenodo.2573505
50. Roetteler, M., Naehrig, M., Svore, K.M., Lauter, K.E.: Quantum resource estimates for computing elliptic curve discrete logarithms. In: ASIACRYPT (2). Lecture Notes in Computer Science, vol. 10625, pp. 241–270. Springer (2017). https://doi.org/10.1007/978-3-319-70697-9_9

51. Sendrier, N.: Decoding one out of many. In: PQCrypto. Lecture Notes in Computer Science, vol. 7071, pp. 51–67. Springer (2011). https://doi.org/10.1007/978-3-642-25405-5_4
52. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: FOCS. pp. 124–134. IEEE Computer Society (1994). https://doi.org/10.1109/SFCS.1994.365700
53. Stern, J.: A method for finding codewords of small weight. In: Coding Theory and Applications. Lecture Notes in Computer Science, vol. 388, pp. 106–113. Springer (1988). https://doi.org/10.1007/BFB0019850
54. Wiedemann, D.H.: Solving sparse linear equations over finite fields. IEEE Trans. Inf. Theory **32**(1), 54–62 (1986). https://doi.org/10.1109/TIT.1986.1057137

A Quantum Circuit Components

The circuits that we construct in this paper are *classical reversible* circuits, containing only Toffoli (CCX), CNOT (CX) and NOT (X) gates. In this section, we introduce some useful building blocks for these circuits, and give their associated cost. All of them have been implemented.

A.1 Fan-in, Fan-out and Reverse

Lemma 10 (Fan-in / fan-out). *For any n, there exists a* fan-in *circuit operating on $n+1$ bits which maps:*

$$(v_0, \ldots, v_{n-1}, b) \mapsto (v_0, \ldots, v_{n-1}, b + (\sum_i v_i)) \ ,$$

and a fan-out *circuit operating on $n+1$ bits which maps:*

$$(b, 0, \ldots, 0) \mapsto (b, b, \ldots, b) \ .$$

The fan-in circuit uses $2n-1$ CX gates and has depth $\leq 2\lceil \log_2 n \rceil + 1$. The fan-out circuit uses n CX gates and has depth $\leq \lceil \log_2 n \rceil + 1$.

Proof. In the fan-out case, we first XOR b to the bit at position 0. Then for $i = 0$ to $\lceil \log_2 n \rceil - 1$, we XOR the bits at positions $0, \ldots, 2^i - 1$ to the bits at positions $2^i, \ldots, 2^{i+1} - 1$, doubling the number of bits that have been seen. Each new bit costs us one CX gate, so the total is n CX gates. Each new step has depth one, so the total is $\lceil \log_2 n \rceil + 1$ depth.

The fan-in circuit is very similar, except that we first need to aggregate all values in the bit v_{n-1}, using a tree of depth $\lceil \log_2 n \rceil$ and $n-1$ CXs, then we XOR the result to b, and then we do the tree again to return $v_0, \ldots, v_{n-1}$ to their initial values. □

Lemma 11. *For any constant k, there exists a* controlled-shift *circuit on registers of length n:*

$$\begin{cases} 0, (v_0, \ldots, v_{n-1}) \mapsto 0, (v_0, \ldots, v_{n-1}) \\ 1, (v_0, \ldots, v_{n-1}) \mapsto 1, (v_k, \ldots, v_{n-1}, v_0, \ldots, v_{k-1}) \end{cases} \tag{31}$$

using $4n$ CX and $3n$ CCX gates, $\mathcal{O}(n)$ ancilla qubits and depth $\mathcal{O}(\log n)$.

Proof. The circuit has the following steps:

- Fan-out the control bit (depth $\mathcal{O}(\log n)$, n CX gates)
- Copy the bits v_i into a new ancilla register, using Toffoli gates with the copies of the control (depth 1, n Toffoli gates)
- Erase the input bits v_i by performing the reverse operation (depth 1, n CCX gates)
- Perform a controlled swap of the input and output bits. A controlled swap between bit b and b' is implemented as $CX(b, b'), CCX(c, b', b), CX(b, b')$ (depth 3, n CCX and $2n$ CX gates)
- Reverse the fan-out (depth $\mathcal{O}(\log n)$, n CX gates)

In total we have used $\mathcal{O}(n)$ gates. The depth is dominated by the logarithmic depth of the fan-out circuit. □

The following is useful for manipulating the polynomials in the Berlekamp-Massey algorithm. In an input list that represents the polynomial $C(X) = c_0 + c_1X + \ldots + c_dX^d$, where $c_0 = 1$, we want to obtain $\mathsf{Reverse}(C)(X) = c_d + c_{d-1}X + \ldots + c_0X^d$. The main issue here is that d is variable.

Lemma 12 (Reversion circuit). *There exists a reversible* reversion circuit *that given a register of n bits, reverses the order of the bits without taking into account the trailing zeroes. The circuit contains $\mathcal{O}(n \log n)$ gates, uses $\mathcal{O}(n)$ ancilla qubits and has depth $\mathcal{O}(n)$.*

Proof. Let $x_0, \ldots, x_{n-1}$ be the input bits. We start by computing the degree $d \leq n - 1$. For this operation we start by negating all input bits. We initialize a register of n bits to zero, and we will write down a sequence of bits $b_0, \ldots, b_{n-1}$ starting from the position $n - 1$ downwards, with definition: $b_i = b_{i+1}\bar{x_i}$. That is, b_i indicates whether all $\bar{x_j}$ for $j \geq i$ are one, i.e., all x_j are zero. At this point we have used $\mathcal{O}(n)$ depth and gates.

We have then: $n+1-d = \sum_i b_i$. This sum can be computed in time $\mathcal{O}(n \log n)$ using a counter that iterates over the b_i, and in better depth using multiple counters and adding their results. This does not dominate the cost.

After obtaining the degree, we reverse the entire n-bit list using swaps with fixed positions. Then, we shift the list by $n + 1 - d$ positions left. This is done using $\mathcal{O}(\log n)$ successive controlled-shift circuits, so a total of $\mathcal{O}(n \log n)$ gates and $\mathcal{O}\big(\log^2 n\big)$ depth (the depth does not dominate).

Since a polynomial and its reverse have the same degree, we can uncompute it. This completes the circuit definition. □

A.2 Circulant Matrix-Vector Multiplication

Some of our circuits require the implementation of an *out-of-place* multiplication of a binary vector $\mathbf{x} \in \mathbb{F}_2^n$ by a *constant* matrix $\mathbf{H} \in \mathbb{F}_2^{m \times n}$:

$$|\mathbf{x}\rangle\,|\mathbf{y}\rangle \xmapsto{\mathsf{MultConstant}_{\mathbf{H}}} |\mathbf{x}\rangle\,|\mathbf{y} + \mathbf{H}\mathbf{x}\rangle \quad .$$

If the matrix has no particular structure, matrix-vector product can be computed naively.

Lemma 13 (Matrix multiplication). *There exists a reversible circuit for* $\mathsf{MultConstant}_{\mathbf{H}}$ *using* $\leq mn$ *CX gates, depth* $\max(m,n)$ *and no ancilla qubits.*

Proof. Let us note: $\mathbf{x} := (x_0, \ldots, x_{n-1})^T$, $\mathbf{y} := (y_0, \ldots, y_{m-1})^T$ and $\mathbf{H} := (h_{ij})_{0 \leq i \leq m-1, 0 \leq j \leq n-1}$. Since the h_{ij} are binary and constant, we just need to apply a CX over (x_j, y_i) for any pair (i,j) such that $h_{ij} = 1$. We can do this efficiently in depth by considering disjoint pairs. Suppose that $n \geq m$, then we first loop over $i \in \{0, \ldots, n-1\}$, then over $j \in \{0, \ldots, m-1\}$, and consider the pair $(j, (i+j) \mod n)$. Clearly, for constant i, all the pairs $(j, (i+j) \mod n)$ are disjoint, meaning that the CXs can be applied in a single layer. The full circuit has n layers. If $n > m$, then we can exchange the roles of i and j. □

Notice that we can expect the CX count to be twice as low if $\mathbf{H}$ is selected uniformly at random.

Karatsuba Multiplication of Polynomials. It is well known that multiplying a vector in $\mathbb{F}_2^n$ by a binary circulant matrix is equivalent to multiplying polynomials in $\mathbb{F}_2[X]$. This operation is an important building block which has been very well-studied classically, and its gate count has been optimized for large polynomials of fixed degree [10]. However, the existing classical circuits are in essence non-reversible. In the quantum setting, van Hoof [33] gave a method for Karatsuba multiplication of such polynomials which decreased the CCX gate count but with a $\mathcal{O}(n^2)$ CX count asymptotically.

In our setting, the operation that we need to implement is a multiplication by a *constant* polynomial $P(X) = p_0 + p_1 X + \ldots + p_{n-1} X^{n-1} \in \mathbb{F}_2^{n-1}[X]$:

$$\begin{cases} \mathbb{F}_2^{n-1}[X] \to \mathbb{F}_2^{2n-2}[X] \\ A(X), B(X) \xmapsto{\mathsf{PolMult}_P} A(X), P(X)A(X) + B(X) \end{cases} \tag{32}$$

In particular, it can be noticed that $\mathsf{PolMult}_P$ is a linear mapping over $\mathbb{F}_2$. We would like to use as few nonlinear operations as necessary, and therefore, to implement this circuit using CXs only. We use the approach of Gidney [29] for space-efficient Karatsuba multiplication, which we adapt to our case as follows.

Lemma 14 (Adaptation of [29]). *Given any polynomial* $P(X) \in \mathbb{F}_2^{n-1}[X]$, *there exists a linear circuit implementing* $\mathsf{PolMult}_P$ *using* $\mathcal{O}(n^{\log_2 3})$ *CX gates, with depth* $\mathcal{O}(n^{\log_2 3})$, *without ancilla qubits.*

Proof. We prove the lemma when $n = 2^k$ is a power of 2 by induction over k, by constructing the circuit recursively.

For k small enough, e.g., $k = 6$ ($n = 64$), we use a naive multiplication circuit. Consider a value $k > 6$ and let: $P(X) := P_1(X) + X^n P_2(X)$ and $A(X) := A_1(X) + X^n A_2(X)$ where A_1, A_2, P_1, P_2 are of degree $n = 2^k$. We rewrite:

$$\begin{aligned} &(A_1(X) + X^n A_2(X))(P_1(X) + X^n P_2(X)) \\ &\quad = A_1 P_1 (1 + X^n) + X^n A_2 P_2 (1 + X^n) + X^n ((A_1 + A_2)(P_1 + P_2)) \ . \end{aligned} \tag{33}$$

In the following, we will use three multiplications by a constant degree-n polynomial, and $12n$ additional CX gates. Let W_1, W_2, W_3, W_4 be the 4 n-bit parts of the output register, which start by containing a polynomial $B = B_1 + X^n B_2 + X^{2n} B_3 + X^{3n} B_4$ which *should not* be modified.

We perform the following operations:

1. $(W_1, W_2, W_3) \leftarrow (W_1, W_2, W_3) + A_1 P_1 (1 + X^n)$
   We cannot first compute $A_1 P_1$ and then multiply by $1 + X^n$, as this would modify the polynomial B. Multiplying by $1 + X^n$ is a sequence of n CX gates. So, we first perform these n gates in reverse on (W_1, W_2, W_3), then we call the Karatsuba circuit for P_1 and A_1, and write the output in (W_1, W_2). Then we perform the CX gates in order.
2. $(W_2, W_3, W_4) \leftarrow (W_2, W_3, W_4) + X^n A_2 P_2 (1 + X^n)$
   Following the same principle, we first perform the n CX gates which are the reverse of multiplying by $1 + X^n$ on (W_2, W_3). Then we call the Karatsuba circuit for P_2 and A_2, write the output in (W_2, W_3). Then we perform the CX gates in order.
3. $(W_2, W_3, W_4) \leftarrow (W_2, W_3, W_4) + X^n((A_1 + A_2)(P_1 + P_2))$
   We CX A_2 to A_1 using n gates, then we call the Karatsuba circuit for $(P_1 + P_2)$. Then we CX A_2 to A_1 again to restore its state.

The correctness follows from Equation 33. As for the complexity analysis, we can see that no ancilla qubits are required throughout the algorithm. The gate count and depth are respectively:

$$\begin{cases} G(2^k) = 3G(2^{k-1}) + 6 \cdot 2^k \\ D(2^k) = 3D(2^{k-1}) + 6 \end{cases} \tag{34}$$

In practice, we consider polynomials of degree $2^k \cdot u$ and set a threshold for naive multiplication at u for a constant u to optimize later. Like matrices, multiplication of degree-u polynomials can be done using u^2 CX gates and in depth u. This makes these quantities bounded by:

$$\begin{aligned} G(2^k u) &= 3^k G(u) + 6(2^k + 3 \cdot 2^{k-1} + \ldots + 3^{k-1} \cdot 2)u \\ &\leq 3^k u^2 + 6 \cdot 3^k \cdot 2 = 3^k(u^2 + 12) \ . \\ D(2^k u) &= 3^k D(u) + 6(1 + 3 + \ldots + 3^{k-1}) \leq 3^k(u + 3) \ . \end{aligned}$$

Choosing u a constant, these formulas yield the expected asymptotic complexities. □

As a corollary, we obtain a better circuit for multiplication by a constant circulant matrix. The circuit is more efficient in CX count, but less efficient in depth. A trade-off can be achieved by setting appropriately the level at which one performs naive polynomial multiplication.

Corollary 1. *For any k and u, there exists a circuit for multiplying a binary vector of length $2^k u$ by a constant circulant matrix of dimension $2^k u$, using $\leq 3^k(2u^2 + 24) + 2^{k+1}u$ CNOT gates and depth $\leq 3^k(u + 3) + 2$. It uses $2^{k+2}u$ qubits.*

Proof. The circuit simply computes a polynomial multiplication, then XORs the two halves of the output polynomial into the output vector, then uncomputes the polynomial multiplication. □